IFW

UNITED STATES PATENT AND TRADEMARK OFFICE

OIPE JUL 0 8 2008 PATENT & TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

CANCELLED

OIPE JUN 0 8 2008 PATENT & TRADEMARK OFFICE

NOTICE OF ALLOWANCE AND FEE(S) DUE

7590 06/12/2008

Stattler Johansen & Adeli
P O Box 51860
Palo Alto, CA 94303-0728

EXAMINER	
COLAN, GIOVANNA B	
ART UNIT	PAPER NUMBER
2162	

DATE MAILED: 06/12/2008

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/534,627	01/03/2006	Steven David Lavine	TRAN.P0001	8549

TITLE OF INVENTION: SEARCH METHOD AND SYSTEM AND SYSTEM USING THE SAME

APPLN. TYPE	SMALL ENTITY	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	YES	$720	$300	$0	$1020	09/12/2008

THE APPLICATION IDENTIFIED ABOVE HAS BEEN EXAMINED AND IS ALLOWED FOR ISSUANCE AS A PATENT. PROSECUTION ON THE MERITS IS CLOSED. THIS NOTICE OF ALLOWANCE IS NOT A GRANT OF PATENT RIGHTS. THIS APPLICATION IS SUBJECT TO WITHDRAWAL FROM ISSUE AT THE INITIATIVE OF THE OFFICE OR UPON PETITION BY THE APPLICANT. SEE 37 CFR 1.313 AND MPEP 1308.

THE ISSUE FEE AND PUBLICATION FEE (IF REQUIRED) MUST BE PAID WITHIN THREE MONTHS FROM THE MAILING DATE OF THIS NOTICE OR THIS APPLICATION SHALL BE REGARDED AS ABANDONED. THIS STATUTORY PERIOD CANNOT BE EXTENDED. SEE 35 U.S.C. 151. THE ISSUE FEE DUE INDICATED ABOVE DOES NOT REFLECT A CREDIT FOR ANY PREVIOUSLY PAID ISSUE FEE IN THIS APPLICATION. IF AN ISSUE FEE HAS PREVIOUSLY BEEN PAID IN THIS APPLICATION (AS SHOWN ABOVE), THE RETURN OF PART B OF THIS FORM WILL BE CONSIDERED A REQUEST TO REAPPLY THE PREVIOUSLY PAID ISSUE FEE TOWARD THE ISSUE FEE NOW DUE.

HOW TO REPLY TO THIS NOTICE:

I. Review the SMALL ENTITY status shown above.

If the SMALL ENTITY is shown as YES, verify your current SMALL ENTITY status:

A. If the status is the same, pay the TOTAL FEE(S) DUE shown above.

B. If the status above is to be removed, check box 5b on Part B - Fee(s) Transmittal and pay the PUBLICATION FEE (if required) and twice the amount of the ISSUE FEE shown above, or

If the SMALL ENTITY is shown as NO:

A. Pay TOTAL FEE(S) DUE shown above, or

B. If applicant claimed SMALL ENTITY status before, or is now claiming SMALL ENTITY status, check box 5a on Part B - Fee(s) Transmittal and pay the PUBLICATION FEE (if required) and 1/2 the ISSUE FEE shown above.

II. PART B - FEE(S) TRANSMITTAL, or its equivalent, must be completed and returned to the United States Patent and Trademark Office (USPTO) with your ISSUE FEE and PUBLICATION FEE (if required). If you are charging the fee(s) to your deposit account, section "4b" of Part B - Fee(s) Transmittal should be completed and an extra copy of the form should be submitted. If an equivalent of Part B is filed, a request to reapply a previously paid issue fee must be clearly made, and delays in processing may occur due to the difficulty in recognizing the paper as an equivalent of Part B.

III. All communications regarding this application must give the application number. Please direct all communications prior to issuance to Mail Stop ISSUE FEE unless advised to the contrary.

IMPORTANT REMINDER: Utility patents issuing on applications filed on or after Dec. 12, 1980 may require payment of maintenance fees. It is patentee's responsibility to ensure timely payment of maintenance fees when due.

PART B - FEE(S) TRANSMITTAL

Complete and send this form, together with applicable fee(s), to: Mail **Mail Stop ISSUE FEE**
Commissioner for Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450
or Fax (571)-273-2885

INSTRUCTIONS: This form should be used for transmitting the ISSUE FEE and PUBLICATION FEE (if required). Blocks 1 through 5 should be completed where appropriate. All further correspondence including the Patent, advance orders and notification of maintenance fees will be mailed to the current correspondence address as indicated unless corrected below or directed otherwise in Block 1, by (a) specifying a new correspondence address; and/or (b) indicating a separate "FEE ADDRESS" for maintenance fee notifications.

CURRENT CORRESPONDENCE ADDRESS (Note: Use Block 1 for any change of address)

7590 06/12/2008

Stattler Johansen & Adeli
P O Box 51860
Palo Alto, CA 94303-0728

Note: A certificate of mailing can only be used for domestic mailings of the Fee(s) Transmittal. This certificate cannot be used for any other accompanying papers. Each additional paper, such as an assignment or formal drawing, must have its own certificate of mailing or transmission.

Certificate of Mailing or Transmission
I hereby certify that this Fee(s) Transmittal is being deposited with the United States Postal Service with sufficient postage for first class mail in an envelope addressed to the Mail Stop ISSUE FEE address above, or being facsimile transmitted to the USPTO (571) 273-2885, on the date indicated below.

(Depositor's name)
(Signature)
(Date)

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/534,627	01/03/2006	Steven David Lavine	TRAN.P0001	8549

TITLE OF INVENTION: SEARCH METHOD AND SYSTEM AND SYSTEM USING THE SAME

APPLN. TYPE	SMALL ENTITY	ISSUE FEE DUE	PUBLICATION FEE DUE	PREV. PAID ISSUE FEE	TOTAL FEE(S) DUE	DATE DUE
nonprovisional	YES	$720	$300	$0	$1020	09/12/2008

EXAMINER	ART UNIT	CLASS-SUBCLASS
COLAN, GIOVANNA B	2162	707-103000

1. Change of correspondence address or indication of "Fee Address" (37 CFR 1.363).
   - ☐ Change of correspondence address (or Change of Correspondence Address form PTO/SB/122) attached.
   - ☐ "Fee Address" indication (or "Fee Address" Indication form PTO/SB/47; Rev 03-02 or more recent) attached. **Use of a Customer Number is required.**

2. For printing on the patent front page, list
   (1) the names of up to 3 registered patent attorneys or agents OR, alternatively,
   (2) the name of a single firm (having as a member a registered attorney or agent) and the names of up to 2 registered patent attorneys or agents. If no name is listed, no name will be printed.
   1 ____________
   2 ____________
   3 ____________

3. ASSIGNEE NAME AND RESIDENCE DATA TO BE PRINTED ON THE PATENT (print or type)
PLEASE NOTE: Unless an assignee is identified below, no assignee data will appear on the patent. If an assignee is identified below, the document has been filed for recordation as set forth in 37 CFR 3.11. Completion of this form is NOT a substitute for filing an assignment.
(A) NAME OF ASSIGNEE (B) RESIDENCE: (CITY and STATE OR COUNTRY)

Please check the appropriate assignee category or categories (will not be printed on the patent) : ☐ Individual ☐ Corporation or other private group entity ☐ Government

4a. The following fee(s) are submitted:
- ☐ Issue Fee
- ☐ Publication Fee (No small entity discount permitted)
- ☐ Advance Order - # of Copies ____________

4b. Payment of Fee(s): **(Please first reapply any previously paid issue fee shown above)**
- ☐ A check is enclosed.
- ☐ Payment by credit card. Form PTO-2038 is attached.
- ☐ The Director is hereby authorized to charge the required fee(s), any deficiency, or credit any overpayment, to Deposit Account Number ____________ (enclose an extra copy of this form).

5. **Change in Entity Status** (from status indicated above)
☐ a. Applicant claims SMALL ENTITY status. See 37 CFR 1.27. ☐ b. Applicant is no longer claiming SMALL ENTITY status. See 37 CFR 1.27(g)(2).

NOTE: The Issue Fee and Publication Fee (if required) will not be accepted from anyone other than the applicant; a registered attorney or agent; or the assignee or other party in interest as shown by the records of the United States Patent and Trademark Office.

Authorized Signature ____________ Date ____________

Typed or printed name ____________ Registration No. ____________

This collection of information is required by 37 CFR 1.311. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 12 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, Virginia 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, Virginia 22313-1450.

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/534,627	01/03/2006	Steven David Lavine	TRAN.P0001	8549

7590 06/12/2008

Stattler Johansen & Adeli
P O Box 51860
Palo Alto, CA 94303-0728

EXAMINER	
COLAN, GIOVANNA B	
ART UNIT	PAPER NUMBER
2162	

DATE MAILED: 06/12/2008

Determination of Patent Term Adjustment under 35 U.S.C. 154 (b)
(application filed on or after May 29, 2000)

The Patent Term Adjustment to date is 0 day(s). If the issue fee is paid on the date that is three months after the mailing date of this notice and the patent issues on the Tuesday before the date that is 28 weeks (six and a half months) after the mailing date of this notice, the Patent Term Adjustment will be 0 day(s).

If a Continued Prosecution Application (CPA) was filed in the above-identified application, the filing date that determines Patent Term Adjustment is the filing date of the most recent CPA.

Applicant will be able to obtain more detailed information by accessing the Patent Application Information Retrieval (PAIR) WEB site (http://pair.uspto.gov).

Any questions regarding the Patent Term Extension or Adjustment determination should be directed to the Office of Patent Legal Administration at (571)-272-7702. Questions relating to issue and publication fee payments should be directed to the Customer Service Center of the Office of Patent Publication at 1-(888)-786-0101 or (571)-272-4200.

Notice of Allowability	**Application No.** 10/534,627	**Applicant(s)** LAVINE, STEVEN DAVID
	Examiner GIOVANNA COLAN	**Art Unit** 2162

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address--
All claims being allowable, PROSECUTION ON THE MERITS IS (OR REMAINS) CLOSED in this application. If not included herewith (or previously mailed), a Notice of Allowance (PTOL-85) or other appropriate communication will be mailed in due course. **THIS NOTICE OF ALLOWABILITY IS NOT A GRANT OF PATENT RIGHTS.** This application is subject to withdrawal from issue at the initiative of the Office or upon petition by the applicant. See 37 CFR 1.313 and MPEP 1308.

1. ☒ This communication is responsive to *amendment filed 02/25/2008*.

2. ☒ The allowed claim(s) is/are *1 – 3, 6 – 8, 11 – 16, 18 – 19, 22 – 23, and 68 – 85*.

3. ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
   a) ☐ All b) ☐ Some* c) ☐ None of the:
   1. ☐ Certified copies of the priority documents have been received.
   2. ☐ Certified copies of the priority documents have been received in Application No. ____ .
   3. ☐ Copies of the certified copies of the priority documents have been received in this national stage application from the International Bureau (PCT Rule 17.2(a)).

   * Certified copies not received: ____.

Applicant has THREE MONTHS FROM THE "MAILING DATE" of this communication to file a reply complying with the requirements noted below. Failure to timely comply will result in ABANDONMENT of this application.
THIS THREE-MONTH PERIOD IS NOT EXTENDABLE.

4. ☐ A SUBSTITUTE OATH OR DECLARATION must be submitted. Note the attached EXAMINER'S AMENDMENT or NOTICE OF INFORMAL PATENT APPLICATION (PTO-152) which gives reason(s) why the oath or declaration is deficient.

5. ☐ CORRECTED DRAWINGS (as "replacement sheets") must be submitted.
   (a) ☐ including changes required by the Notice of Draftsperson's Patent Drawing Review (PTO-948) attached
   1) ☐ hereto or 2) ☐ to Paper No./Mail Date ____.
   (b) ☐ including changes required by the attached Examiner's Amendment / Comment or in the Office action of Paper No./Mail Date ____.

Identifying indicia such as the application number (see 37 CFR 1.84(c)) should be written on the drawings in the front (not the back) of each sheet. Replacement sheet(s) should be labeled as such in the header according to 37 CFR 1.121(d).

6. ☐ DEPOSIT OF and/or INFORMATION about the deposit of BIOLOGICAL MATERIAL must be submitted. Note the attached Examiner's comment regarding REQUIREMENT FOR THE DEPOSIT OF BIOLOGICAL MATERIAL.

Attachment(s)

1. ☒ Notice of References Cited (PTO-892)	5. ☐ Notice of Informal Patent Application
2. ☐ Notice of Draftperson's Patent Drawing Review (PTO-948)	6. ☐ Interview Summary (PTO-413), Paper No./Mail Date ____ .
3. ☐ Information Disclosure Statements (PTO/SB/08), Paper No./Mail Date ____	7. ☒ Examiner's Amendment/Comment
4. ☐ Examiner's Comment Regarding Requirement for Deposit of Biological Material	8. ☒ Examiner's Statement of Reasons for Allowance
	9. ☐ Other ____.

/Jean M Corrielus/
Primary Examiner, Art Unit 2162

EXAMINER'S AMENDMENT

1. An examiner's amendment to the record appears below. Should the changes and/or additions be unacceptable to applicant, an amendment may be filed as provided by 37 CFR 1.312. To ensure consideration of such an amendment, it MUST be submitted no later than the payment of the issue fee.

2. Authorization for this examiner's amendment was given in a telephone interview with Gregory Suh on June 04, 2008.

3. The application has been amended as follows:

In the claims:

Please amend claims 1, 6, and 13 as follows:

1. A method for searching a content database stored in computer storage, the content database including a plurality of records each containing multiple fields of information, the method comprising the steps of:

maintaining a structure database in computer storage in which each record is parsed into one or more record categories, each record category having zero or more sub-categories and one or more fields of information, the structure database containing, for each record category, information defining ~~the~~ a data structure of the record category;

receiving a search query comprising one or more query categories, each query category comprising zero or more sub-categories and one or more selections from a user;

determining for each query category, ~~the~~ a data structure of the query category based on the data structure of a corresponding record category;

for each of one or more records, performing a correlation between the data structure of each query category and the data structure of the corresponding record category to produce a relevance value for the record, wherein performing the correlation comprises:

for each data structure of a query category, generating a selection tree comprising a node representing the query category, sub-nodes representing the sub-categories and selections, and weights for each node and sub-node assigned based on the selections from the user, and

for each data structure of the corresponding record category, generating a data tree comprising a node representing the record category, sub-nodes representing the sub-categories and fields of information, and weights for each node and sub-node assigned based on the level of the node or sub-node in the data tree or based on the selections from the user, and using a correlation algorithm to correlate the weights of the data tree with the weights of the selection tree to produce a relevance value for the corresponding record category; and

as a response to the search query, selecting records in the content database based upon the relevance values for the one or more records.

6. A system for searching a content database stored in computer storage, the content database including a plurality of records each containing multiple fields of information, the system comprising:

a structure database in computer storage in which each record is parsed into one or more record categories, each record category having zero or more sub-categories and one or more fields of information, the structure database containing, for each record category, information defining ~~the~~ a data structure of the record category;

a receiver for receiving a search query comprising one or more query categories, each query category comprising zero or more sub-categories and one or more selections from a user;

a determining device for determining, for each query category, ~~the~~ a data structure of the query category based on the data structure of a corresponding record category;

a correlation device for performing, for each of one or more records, a correlation between the data structure of each query category and the data structure of the corresponding record category to produce a relevance value for the record, wherein performing the correlation comprises:

for each data structure of a query category, generating a selection tree comprising a node representing the query category, sub-nodes representing the sub- categories and selections, and weights for each node and sub-node assigned based on the selections from the user, and for each data structure of the corresponding record category, generating a data tree comprising a node representing the record category, sub-nodes representing the sub-categories and fields of information, and weights for each node and sub-node assigned based on the level of the node or sub-node in the data tree or based on the selections from the user, and using a correlation algorithm to correlate the weights of the data tree with the weights of the selection tree to produce a relevance value for the corresponding record category; and

a response unit for responding to the search query by selecting and providing records in the content database based upon the relevance values for the one or more records.

13. In an online user forum of the type permitting communication among a plurality of users and also permitting users to post information content for access by users, the improvement comprising a reputation module storing a reputation rating for a user in association with information content, a user's reputation being a function of the degree of his participation in the forum, said reputation module being included within a system for searching a content database stored in computer storage, the content database including a plurality of records each containing multiple fields of information, the system further comprising:

a structure database in computer storage in which each record is parsed into one or more record categories, each record category having zero or more sub-categories and one or more fields of information, the structure database containing, for each record category, information defining ~~the~~ a data structure of the record category;

a receiver for receiving a search query comprising one or more query categories, each query category comprising zero or more sub-categories and one or more selections from a user;

a determining device for determining, for each query category, ~~the~~ a data structure of the query category based on the data structure of a corresponding record category;

a correlation device for performing, for each of one or more records, a correlation between the data structure of each query category and the data structure of the corresponding record category to produce a relevance value for the record, wherein performing the correlation comprises:

for each data structure of a query category, generating a selection tree comprising a node representing the query category, sub-nodes representing the sub-categories and selections, and weights for each node and sub-node assigned based on the selections from the user, and

for each data structure of the corresponding record category, generating a data tree comprising a node representing the record category, sub-nodes representing the sub-categories and fields of information, and weights for each node and sub-node assigned based on the level of the node or sub-node in the data tree or based on the selections from the user, and using a correlation algorithm to correlate the weights of the data tree with the weights of the selection tree to produce a relevance value for the corresponding record category; and

a response unit for responding to the search query by selecting and providing records in the content database based upon the relevance values for the one or more records.

REASONS FOR ALLOWANCE

4. Claims 1 – 3, 6 – 8, 11 – 16, 18 – 19, 22 – 23, and 68 – 85 are allowable in light of the applicant's arguments and in light of the prior art made of record.

Reason for Indicating Allowable Subject Matter

5. The following is an examiner's statement of reasons for allowance: Upon searching a variety of databases, the examiner respectfully submits that "for each of one or more records, performing a correlation between the data structure of each query category and the data structure of the corresponding record category to produce a relevance value for the record, wherein performing the correlation comprises: for each data structure of a query category, generating a selection tree comprising a node representing the query category, sub-nodes representing the sub-categories and selections, and weights for each node and sub-node assigned based on the selections from the user, and for each data structure of the corresponding record category, generating a data tree comprising a node representing the record category, sub-nodes representing the sub-categories and fields of information, and weights for each node

and sub-node assigned based on the level of the node or sub-node in the data tree or based on the selections from the user, and using a correlation algorithm to correlate the weights of the data tree with the weights of the selection tree to produce a relevance value for the corresponding record category" in conjunction with all other limitations of the dependent and independent claims are not taught nor suggested by the prior art of record. Therefore, all pending claims 1 – 3, 6 – 8, 11 – 16, 18 – 19, 22 – 23, and 68 – 85 are hereby allowed.

9. Any comments considered necessary by applicant must be submitted no later than the payment of the issues fee. Such submissions should be clearly labeled "Comments on Statement of Reasons for Allowance."

Points Of Contact

Any inquiry concerning this communication or earlier communications from the examiner should be directed to GIOVANNA COLAN whose telephone number is (571)272-2752. The examiner can normally be reached on 8:30 am - 5:00 pm.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, John Breene can be reached on (571) 272-4107. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

Giovanna Colan
Examiner
Art Unit 2162
June 5, 2008
/Jean M Corrielus/
Primary Examiner, Art Unit 2162

Notice of References Cited	Application/Control No. 10/534,627	Applicant(s)/Patent Under Reexamination LAVINE, STEVEN DAVID	
	Examiner GIOVANNA COLAN	Art Unit 2162	Page 1 of 1

U.S. PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
*	A	US-5,642,502 A	06-1997	Driscoll, James R.	707/5
*	B	US-6,098,066 A	08-2000	Snow et al.	707/3
*	C	US-2001/0007987 A1	07-2001	Igata, Nobuyuki	707/3
*	D	US-2001/0046677 A1	11-2001	Liu et al.	435/6
*	E	US-2002/0055932 A1	05-2002	Wheeler et al.	707/104.1
*	F	US-2002/0111847 A1	08-2002	Smith, James R. II	705/10
*	G	US-2002/0147711 A1	10-2002	Hattori et al.	707/3
*	H	US-6,691,108 B2	02-2004	Li, Wen-Syan	707/3
*	I	US-7,181,438 B1	02-2007	Szabo, Andrew	707/2
*	J	US-7,296,009 B1	11-2007	Jiang et al.	707/3
	K	US-			
	L	US-			
	M	US-			

FOREIGN PATENT DOCUMENTS

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

NON-PATENT DOCUMENTS

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	"Proximal Nodes: A Model to Query Document Databases by Content and Structure"; Gonzalo Navarro and Ricardo Baeza-Yates; University of Chile; ACM; October 1997.
	V	"Grammatical Tree Matching"; Pekka Kilpelainen and Heikki Mannila; University of Helsinki; Finland; 1992.
	W	
	X	

*A copy of this reference is not being furnished with this Office action. (See MPEP § 707.05(a).)
Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

Proximal Nodes: A Model to Query Document Databases by Content and Structure

GONZALO NAVARRO and RICARDO BAEZA-YATES
University of Chile

A model to query document databases by both their content and structure is presented. The goal is to obtain a query language that is expressive in practice while being efficiently implementable, features not present at the same time in previous work. The key ideas of the model are a set-oriented query language based on operations on nearby structure elements of one or more hierarchies, together with content and structural indexing and bottom-up evaluation. The model is evaluated in regard to expressiveness and efficiency, showing that it provides a good trade-off between both goals. Finally, it is shown how to include in the model other media different from text.

Categories and Subject Descriptors: H.1.2 [**Information Systems**]: User/Machine Systems—*human information processing*; H.2.1 [**Database Management**]: Logical Design—*data models*; H.2.2 [**Database Management**]: Physical Design—*access methods*; H.2.3 [**Database Management**]: Languages—*query languages*; H.2.4 [**Database Management**]: Systems—*query processing*; H.3 [**Information Storage and Retrieval**]; I.7.2 [**Text Processing**]: Document Preparation—*format and notation; languages and systems; standards*; I.7.3 [**Text Processing**]: Index Generation

General Terms: Algorithms, Design, Human Factors, Languages, Performance

Additional Key Words and Phrases: Expressivity and efficiency of query languages, hierarchical documents, structured text, text algebras

1. INTRODUCTION

Document databases are deserving more and more attention, due to their multiple applications: digital libraries, office automation, software engineering, automated dictionaries, encyclopedias, etc. [Frakes and Baeza-Yates 1992]. The purpose of a document database is to store documents, structured or not. A document database is composed of two parts: content

This work was supported in part by Grants FONDECYT (Chile) 1940271 and 1950622. A preliminary partial version of this work appeared in the *Proceedings of ACM SIGIR 95*.
Authors' address: Departamento de Ciencias de la Computación, Universidad de Chile, Blanco Encalada 2120, Santiago, Chile; email: {gnavarro; rbaeza}@dcc.uchile.cl.

and (if present) structure. The content is the data itself, while the structure relates different parts of the database by some criterion.

Any information model for a document database should comprise three parts: data, structure, and query language. It must specify how the data are seen (i.e., image formats, character set, etc.), the structuring mechanism (i.e., markup, index structure, etc.), and the query language (i.e., what things can be asked, what the answers are, etc.).

The problem of retrieving information from document databases is normally concentrated on the text part. Text is not a relational table [Codd 1983], in which the information is already formatted and meant to be retrieved by a "key." The information is there, but there is no easy way to extract it. The user must specify what he or she wants, see the results, then reformulate the query, and so on, until satisfied with the answer. Anything that helps the user to find what he or she wants is worth considering.

Traditionally, textual databases are searched by their contents (words, phrases, etc.) or by their structure (e.g., by navigating through a table of contents), but not by both at the same time. Recently, many models have appeared that allow mixing both types of queries.

Mixing contents and structure allows one to pose very powerful queries, being much more expressive than each mechanism by itself. By using a language that integrates both types of queries, the retrieval quality of document databases can be improved. As an example, consider a software development environment with a syntax-directed editor that allows users to search all procedures that use a given global variable without assigning it, all points where a given variable is assigned, or all procedures that invoke a function defined in a given module. Another example is searching in a digital library for books with population statistics graphs, where there are many illustrations regarding birds, or where an oil company is mentioned in a historical context. These queries mix content and structure of the database, and only new models can handle it.

Because of this, we see these models as an evolution from the classical ones. These new models are not fully satisfactory, though. They are not, in general, as mature as the classical ones. Not only do they lack the long process of testing and maturing that traditional models have enjoyed, but also many of them are primitive as software systems, having been implemented mainly as research prototypes.

There are a number of challenges to be faced. On one hand, the "content" of the database is not formatted, but is in natural language form. This means that no traditional model relying on formatted data (e.g., the relational model) or assuming uninterpreted data objects and relying only on their (formatted) attributes (e.g., classical multimedia databases [Bertino et al. 1988]) is powerful enough to represent the wealth of information contained in the text. The information has to be extracted from the text, but not in a rigid way (Sacks-Davis et al. [1994] also argue along the same lines).

On the other hand, there is no consensus on how the structuring model of a database should be. There are a number of possible models, ranging from no structuring at all to complex interrelation networks. Deciding to use a structuring model involves also choosing what kind of queries about structure can be posed.

Finally, there is no consensus on how powerful a model should be. The more powerful the model, the less efficiently it can be implemented. We pay special attention to this expressiveness/efficiency trade-off, since being weak in either of these two aspects makes the model impractical for many applications.

The aim of this article is to present a model for structuring and querying document databases, following the new trend of mixing content and structure in queries. The model is shown to be expressive and efficiently implementable. There is not at this time, to the best of our knowledge, any approach satisfying both goals (see, however, Dao et al. [1996]). We first concentrate on text and later show how to integrate other types of data (e.g., audio, images). These media are becoming more and more common in document databases.

It has been argued that it is better to put a layer integrating a traditional database system with a textual one than to try to design a language comprising all of the features [Sacks-Davis et al. 1994]. Each subsystem focuses on a different part of the query (e.g., integration of an object-oriented database with a structured text engine [Consens and Milo 1994]).

We rely on this approach. We design a language that is focused on exploiting structure- and content-related features. Other features, such as tuples and joins, should be added by integrating this language with another one oriented toward that kind of operation, e.g., a relational database.

We would like to point out what are we *not* covering in this work. First, we do not cover languages that describe document structure, such as SGML [ISO 1986], DSSSL [ISO 1994], SPDL [ISO 1991], and HyTime [ISO 1992]. We do cover structuring models (e.g., hierarchical models). A given structuring model may or may not be expressed using a given language to describe structure.

Second, we concentrate more on querying than on indexing. Although we describe different implementation alternatives for indexing, we consider that updates are less frequent than queries. The efficiency comparison against other models is centered on querying.

Third, we do not describe a fixed query language, but a model into which we show that a number of operations can be expressed. These include widely accepted primitives as well as new ones. The syntax we use for the query operations is not necessarily intended for final users; rather, it is an operational algebra onto which one can map a more user-oriented query language.

Fourth, we do not make an experimental performance comparison between our model and previous work, the main reasons being the following: that for most cases the code is not available; the performance depends heavily on the implementation (we would be comparing algorithms instead of models); the models impose different structures on the text and retrieve different types of elements; and such a study needs a theoretical framework that still does not exist for structured text databases.

Finally, we do not address the important issue of merging structural queries with those involving operations such as relevance ranking (e.g., the sections or titles where the word "computer" is relevant). The reason is that even the simple problem with no relevance considerations is not yet well solved, and an integration between structuring and relevance ranking must be seen as the next goal (see Sacks-Davis et al. [1994] and Arnold-Moore et al. [1995] for some ideas on this problem).

This article is organized as follows. In Section 2 related work is reviewed. In Section 3 our model is presented, in terms of the data model and the operations allowed for queries. In Section 4 the resulting expressiveness is evaluated. In Section 5 we outline the most relevant implementation aspects and formally analyze efficiency. In Section 6 we present a software architecture based on our model. In Section 7 we give an experimental evaluation. In Section 8 we extend our model to include other types of data. In Section 9 our conclusions and future work directions are outlined. A formal syntactic and semantic definition of the model is presented in Appendix A.

Partial earlier versions of this work can be found in Navarro and Baeza-Yates [1995b] and Navarro [1995].

2. RELATED WORK

In this section we briefly review previous approaches to the problem of structuring and querying a textual database. We first mention the traditional ones and then cover novel ideas. For a complete survey, see Baeza-Yates and Navarro [1996].

2.1 Traditional Approaches

There are many classical approaches to the problem of querying a textual database. Some of them are attempts to adapt the relational model [Codd 1983] to include text management [Desai et al. 1986; Stonebraker et al. 1983]; the many traditional models of information retrieval (e.g., the Boolean model, the probabilistic model, the bit-vector model, and the full-text model) [Frakes and Baeza-Yates 1992; Salton and McGill 1983]; hypertext [Conklin 1987] and semantic networks [Hull and King 1987; Tague et al. 1991]; and object-oriented databases [Cattell 1991; Kim and Lochovski 1989] adapted to manage text [Christophides et al. 1994].

None of these approaches satisfy our goal of mixing structure and content in queries (e.g., see Sacks-Davis et al. [1992]). The relational model does not adapt well to the management of text, since it clearly separates structure and content inside the structure, and this is not the case with structured text. Classical information retrieval allows little structuring (normally only plain records and fields). Hypertexts are mostly navigational and oriented toward the (network) structure. Semantic networks model the unformatted data (text, images, etc.) as a set of attributes and facts derivable from them, which is reasonably good for images but very poor for text, which brings a lot of information in its content. Finally, object-oriented databases can express the structure in a natural way, but their facilities for handling text are limited and must be implemented ad hoc. Moreover, path expressions (which are used to extract structural components) are more general than the structure of documents and therefore are less amendable to optimization (e.g., see Consens and Milo [1994] for an example of optimizing path expressions for the particular case of inclusion semantics). Finally, object-oriented databases are record-oriented rather that set-oriented, which is a drawback (since it forces more operational data manipulation, reminiscent of earlier navigational systems) [Date 1995].

Although these models are not powerful enough to extract the information we want from document databases, they do address different problems that pure models oriented toward structure do not generally address (e.g., tuples and joins, attributes, and relevance ranking). We do not compare our model to those, because they deal with different goals.

2.2 Novel Approaches

These approaches are characterized by generally imposing a hierarchical structure on the database, and by mixing queries on content and structure. Although this structuring is simpler than, for example, hypertext, the problem of mixing content and structure is still not satisfactorily solved.

We present a sample of novel models that cover many different approaches:

—*The hybrid model* [Baeza-Yates 1996] models a textual database as a set of documents, which may have *fields* (named areas inside records). These fields do not need to cover all of the text in the document and can nest and overlap. The query language is an algebra over pairs (D, M), where D is a set of documents, and M is a set of *match points* (a text position that matches the searched word or pattern) in those documents. There is a number of operations for obtaining match points, e.g., prefix search and proximity. There are operations for set manipulation of both documents and match points, for restricting matches to only some fields, and for retrieving fields owning some match point. Inclusion relationships can only be queried with respect to a field and a match point; thus the language is not fully compositional. This model can be implemented very efficiently. The original proposal for this model can be found in Baeza-Yates [1994].

—*The PAT expressions model* [Salminen and Tompa 1992] sees only match points, which are used to define *regions*. Regions are defined by pattern-matching expressions that specify what their endpoints look like. Each region represents a set of disjoint *segments*, a segment being a contiguous portion of the text. This allows (1) dynamic definition of regions and (2) translation of all queries on regions to queries on matches. The need to avoid overlapping segments in regions causes a lot of trouble and a lack of orthogonality in the model. This model has been implemented very efficiently [Fawcett 1989]. The cost of this efficiency is in the restrictions, but this cost may for some applications be reasonable.

—*The overlapped lists model* [Clarke et al. 1995] solves the problem of PAT expressions in an elegant way, by allowing overlaps, but not nesting. Each region is a list of (possibly overlapping) segments, originated by textual searches or by named regions (e.g., chapters). The idea is to unify both searches by using an extension of inverted lists, where regions and words are indexed in the same way. The implementation of this model can be as efficient as that of PAT expressions.

—*The lists of references model* [MacLeod 1991] is a general model for structuring and querying textual databases, including hypertext-like linkages, attribute management, and external procedures. The structure of documents is hierarchical (no overlaps), but answers to queries cannot nest (only the top-level elements qualify), and all elements must be from the same type (e.g., only sections or only paragraphs). Answers to queries are seen as lists of "references" (i.e., pointers to the database). This allows integration in an elegant way of answers to queries and hypertext links, since all are seen as lists of references. This model is very powerful and, because of this, hard to implement efficiently. To make the model suitable for comparison, we consider only the portion related to querying structures (even this portion is quite powerful).

—*The parsed strings model* [Gonnet and Tompa 1987] is in fact a structure manipulation language. A context-free grammar is used to express database schemata; that is, the database is structured by giving a grammar to parse its text. The fundamental data structure is the *p-string*, or parsed string, which is composed of a derivation tree plus the underlying text. The manipulation is carried out via a number of operations to transform trees. This approach is extremely powerful and is shown to be relationally complete. However, it is hard to implement efficiently [Blake et al. 1992].

—*The tree-matching model* [Kilpeläinen and Mannila 1993] is a query model relying on a single primitive: tree inclusion. The idea is to model both the structure of the database and the query (a pattern on the structure) as trees, to find an embedding of the pattern into the database that respects the hierarchical relationships between nodes of the pattern. The language is enriched by Prolog-like variables, which can be used to express requirements on equality between parts of the matched substructure and to retrieve another part of the matching subtree, not only the root. The complexity of the algorithms is studied, showing that the only case in which the problem is of polynomial complexity is when no logical variables are used and when the matches have to satisfy the left-to-right ordering in the nodes of the pattern. Even in the polynomial case, the operations have to traverse the whole database structure to find the answers.

3. A NEW MODEL TO QUERY STRUCTURED DOCUMENTS

We now describe our model. We first present the main concepts, then the data model, and finally the query language.

3.1 Main Concepts

In this section we present our general ideas on how a structuring model and a query language can be defined to achieve the goals of efficiency and expressiveness simultaneously. Later, we outline the model following these lines.

Our main goal is to define powerful operations that allow matching on the structure of the database, but that avoid algorithms that match "all-against-all," searching across the whole structure tree (e.g., see Kilpeläinen and Mannila [1992]). A first point is that we want a set-oriented language, because they have been found to be successful in other areas (such as the relational model) and because if we have to extract the whole set of answers it is possible to find algorithms that retrieve the elements at a very low cost per processed element. Since we want to define a fully compositional query language, we consider query expressions as syntax trees, where the nodes represent operations to perform (i.e., operators), and the subtrees represent their operands.

To obtain the set of answers, we avoid a "top-down" approach, where the answers are searched for in the whole structure tree. Rather, we prefer a "bottom-up" strategy. The idea is to find a small set of candidates for the answers quickly and then to eliminate those not meeting the search criterion.

Our solution is an algebra over sets of *structure nodes*. These nodes refer to those of the structure tree of the database. Each one is a structural element, for example, a particular chapter or figure. If they cannot be confused with other types of nodes, we will refer to structure nodes simply as *nodes*.

The operators take sets of nodes and return a set of nodes. These sets of nodes are subsets of the set of all nodes of the structure tree. The only place in which we pose a text-matching query or name a structural component is at the leaves of the query syntax tree. These leaves must be solved with some sort of index and converted to a set of structure nodes. Thereafter, all operators deal with those sets of nodes and produce new sets. Figure 1 shows the main concept, which is refined later to detail the query language and to draw a general software architecture for this model.

With this approach, we use indices to retrieve the nodes that satisfy a text-matching query or that represent a *node type* (e.g., the chapters). Those nodes must be obtained without traversing the whole database.

Once we have converted the leaves of the query syntax tree into sets of structure nodes, all of the other operators take sets of nodes and operate them. Normally, one set will hold the candidates for the result of the operation. Note that we never have to traverse the complete structure when searching.

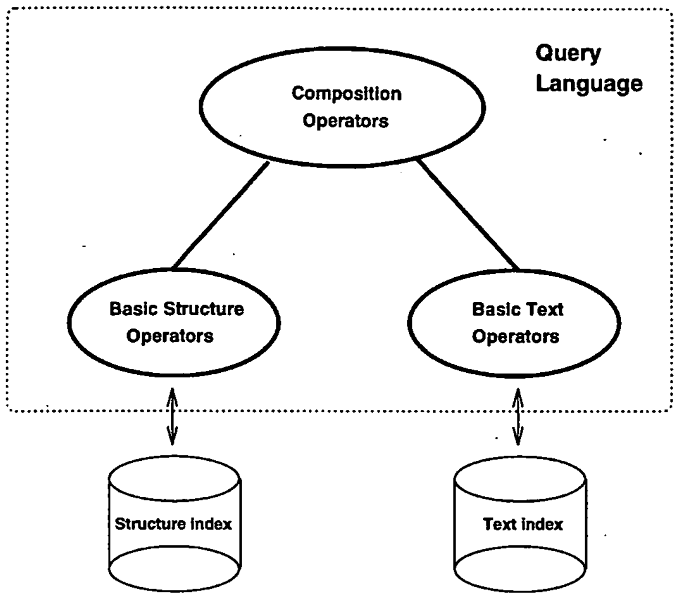

Fig. 1. Initial diagram of how our model operates.

We need still another piece to complete the picture, since at this point the operations performed between sets can be as time-consuming as matching against the database. This piece is the coupling between structure nodes and segments. The segment of a node represents the text it owns; for example, the segment of a chapter includes all of its text. This coupling allows one to use efficient data structures to arrange the nodes by looking at their segments (e.g., forming a tree). In other approaches [Gonnet and Tompa 1987; Kilpeläinen and Mannila 1993], there is a weak binding between nodes and the segment they own in the text, and thus, they need to search the whole tree to find what they need.

In order for this arrangement to be efficient, the operators should only need to access nodes from both sets that are more or less proximal. When this happens, we can obtain the result by traversing both sets of nodes in synchronization.

If we can efficiently convert text-matching expressions and node types into well-arranged sets of nodes, and if all operations can efficiently work with the arranged sets and produce arranged sets, then we will have an efficient implementation. We show later that many interesting operators are in fact of the kind we need; that is, they operate on nearby nodes, and all that they use is the identity of the nodes and their corresponding segment.

Our point is then twofold: first, we must show that a language in which all operations work on nearby nodes can be efficiently implemented; and second, we must show that it is possible to obtain a quite expressive query language by using only this kind of operation. This scheme allows us to have more than one structure hierarchy, if they are independent. We show later that it also allows integration of other media in a natural way.

3.2 Data Model

We now explain how we model the database. We start by redefining more precisely some terms we have been using informally.

A text database is composed of two parts:

(1) *Text*, which is seen as a (long) sequence of symbols (characters, words, etc). Whether this text is stored as it is seen or is filtered to hide markup or uninteresting components is not important for the model, since we use the logical view of the text.

(2) *Structure*, which is organized as a set of independent hierarchies (i.e., disjoint sets of nodes). Each hierarchy has its own types of nodes, and the areas covered by the nodes of different hierarchies can overlap, although this cannot happen inside the same hierarchy. They do not need to cover the whole text. Again, for the model, it is not important how the structure is expressed or extracted from the text.

Filtering out the markup is important, though. The user should not be aware of details about how the structure of the document is internally represented, if it is obtained by parsing, etc. He or she should be able to query the document as it is seen in the output device. If two words are contiguous in the logical view, the user should not be aware that there may be markup between them if, for example, the user is asking for proximity. It may be argued that including the markup in the text allows the user to query about markup by text matching. However, we believe that this must be carried out by the implementation. Any query about markup is probably about structure, and we have a query language for that. The user should not query the structure in such a low-level fashion; he or she should use the content query language to query on the content and the structure query language to query on structure.

The text is static, and the structure built on it is also static. That is, although we allow building, deleting, and modifying new hierarchies, our aim is not to make heavy and continued use of these operations. We are not striving for efficiency in those aspects. Our model of usage is that the text is static; the hierarchies are built on it once (or sparingly); and querying is frequent. The way in which the structure is obtained from the text is not part of the model. It can be obtained by parsing the text, by following markup information, etc.

Each *hierarchy* is a tree of *structure nodes*, or simply *nodes*, and represents an independent way to see the text (e.g., chapters/sections/paragraphs and pages/lines). The root of each hierarchy is a special node that comprises the whole database.

Each hierarchy has a set of *node types* (or "structural components") for its corresponding tree. Examples of node types are page, chapter, and section. The sets of node types of different hierarchies are disjoint.

Each node of a hierarchy belongs to a node type of the hierarchy and has an associated *segment*, which is a pair of numbers representing a contiguous portion of the underlying text. The segment of a node must include the segments of its children in the tree (this inclusion does not need to be strict). We point out that this numbering scheme does not need to be noted by a high-level user, but it can be used to represent more logical-oriented constructions, such as collections of documents.

Any set of disjoint segments can be seen as belonging to a special *text hierarchy*, where the nodes belong to a *text* node type. Thus, the text hierarchy has one node for each possible segment of the text. This is an idealized view that never really appears (only disjoint subsets of nodes can be obtained each time, via pattern-matching queries). Observe that there is no hierarchical relationship between any two nodes of such sets. We say that those sets are *flat*.

The disjointness restriction is in fact not essential, since pattern-matching expressions could perfectly well generate a nested structure. However, this is not normally the case of text pattern-matching languages.

In the Appendix we give a formal definition of the model, and the query syntax and semantics.

3.3 Query Language

In this section we define a query language to operate on the structure defined previously, including queries on the content. We do not intend to define a monolithic, comprehensive query language, since the requirements vary greatly for each application. Including all alternatives in a single query language would make it too complex. Instead, we point out a number of operations that follow our lines (and, hence, can be efficiently implemented).

Each set produced by evaluating a query is a subset of some hierarchy. These subsets are composed of nodes, not subtrees. Although we normally treat them simply as sets, we can consider that the subsets still keep the hierarchical structure they inherit from the complete hierarchy, therefore forming an ordered forest of trees.

We decided not to merge nodes from different hierarchies in a single result for two reasons: first, it is not clear that this would make sense, since hierarchies are different and independent ways to see the same text (e.g., pages or chapters with a figure); second, the implementation is much more efficient if every set is a strict hierarchy. In Clarke et al. [1995] the other choice is selected, i.e., overlaps are allowed in answers, but nested components are not.

Although it is not possible to retrieve subtrees (only nodes), the algebra allows us to select nodes according to their "context" in the structure tree (i.e., what is around them), much like in Kilpeläinen and Mannila [1993]. This language is an operational algebra, not necessarily intended to be accessed by the final user, as the relational algebra is not seen by the users of a relational database. It serves as an intermediate representation of the operations.

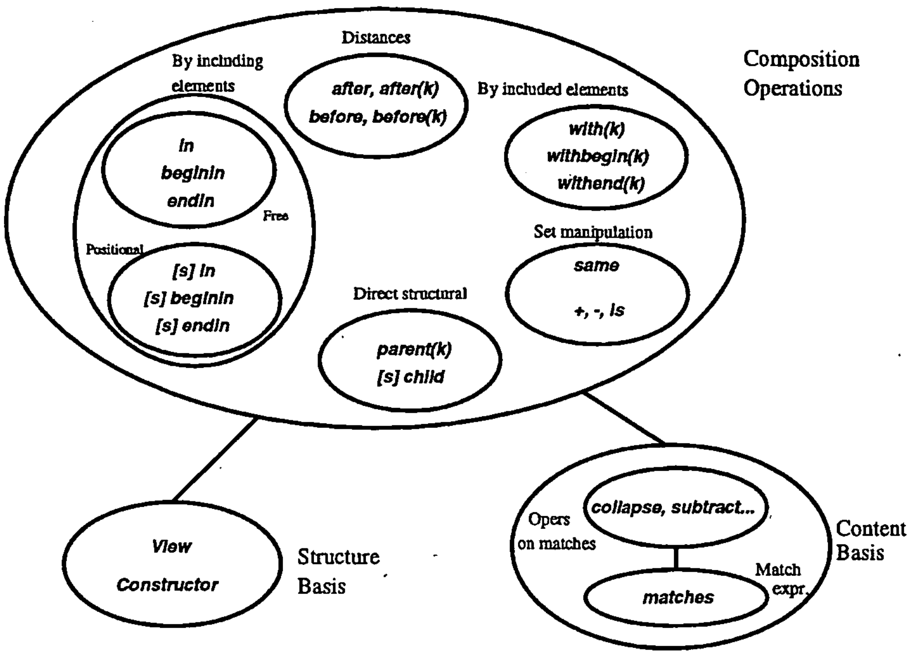

Fig. 2. Possible operations for our model, classified by type.

3.3.1 *Operations.* We list here the operations we consider sufficient for a large set of applications and suitable to be efficiently implemented. As we said before, this set is not exclusive or essential.

Figure 2 shows the schema of the operations. There are basic extraction operators (forming the basis for querying on structure and on contents) and operators to combine results from others, which are classified in a number of groups: those that operate by considering included elements, including elements, or nearby elements; by manipulating sets; and by direct structural relationships.

—*Matching sublanguage:* This is the only one that accesses the text content of the database and is orthogonal to the rest of the language.
  —*Matches:* The matching language generates a set of disjoint segments, which are introduced in the model as belonging to the text hierarchy, as explained earlier. For example, "computer" generates the flat set of all segments of eight letters wherever that word appears in the text. Note that the matching language could allow much more complex expressions (e.g., regular expressions).
  —*Operations on matches:* These are applicable only to subsets of the text hierarchy and make transformations to the segments. Matches and operations on matches are the mechanisms for generating match queries, and we do not restrict our language to any sublanguage for this. See Navarro [1995] for some alternatives. For example, we

propose M **collapse** M', which superimposes both sets of matches, merging them whenever an overlap results.

—*Basic structure operators:* These are the other kind of leaves of the query syntax tree and refer to basic structural components.

—*Name of structural component* ("**Struct**" queries): This is the set of all nodes of the given type. For example, chapter retrieves all chapters in a book.

—*Name of hierarchy* ("**Hierarchy**" queries): This is the set of all nodes of the given hierarchy. For example, Formatting retrieves the whole hierarchy related to formatting aspects. The same effect can be obtained by summing up ("+" operator) all of the node types of the hierarchy.

—*Included-in operators:* These select elements from the first operand that are in some sense included in one of the second operands.

—*Free inclusion:* This selects any included element.

—P **in** Q: This is the set of nodes of P that are included in a node of Q. For example, citation **in** table selects all citations made from inside a table.

—P **beginin/endin** Q: This is the set of nodes of P whose initial/final position is included in a node of Q. For example, page **beginin** formula are the pages that cut the display of a formula (something that we may want to avoid).

—*Positional inclusion:* This selects only those elements included at a given position. In order to define position, only the top-level included elements for each including node are considered.

—$[s]$ P **in** Q: This is the same as **in**, but only qualifies the nodes that descend from a Q-node in a position (from left to right) considered in s. In order to linearize the position, for each node of Q only the top-level nodes of P not disjoint with the Q-node are considered, and those that overlap are discarded, along with their descendants. The language for expressing positions (i.e., values for s) is also independent. We consider that finite unions of $i..j$, $last - i..last - j$, and $i..last - j$ would suffice for most purposes. The range of possible values is $1..last$. For example, [3..5] paragraph **in** page retrieves the third, fourth, and fifth paragraphs from all pages. If paragraphs include other paragraphs, only the top-level ones would be considered, and those partially included in a page would be discarded (along with their subparagraphs).

—$[s]$ P **beginin/endin** Q: This is the same as **beginin/endin**, but uses s as above. For example, [last] page **beginin** chapter selects the last pages of all chapters (which normally are not wholly included in the chapter).

—*Including operators:* These select from the first operand the elements that include in some sense elements from the second operand.

—P **with** (k) Q: This is the set of nodes of P that include at least k nodes of Q. If (k) is not present, we assume 1. For example, section **with** (5) "computer" selects the sections in which the word "computer" appears five or more times.

—P **withbegin/withend** (k) Q: This is the set of nodes of P that include at least k start/end points of nodes of Q. If (k) is not present, we assume 1. For example, chapter **withbegin** (10) page selects chapters with a length of 10 or more pages (assuming each chapter begins at a new page).

—*Direct structure operators:* These select elements from the first operand based on direct structural criteria, that is, by direct parentship in the structure tree corresponding to its hierarchy. Both operands must be from the same hierarchy, which cannot be the text hierarchy.

—$[s]$ P **child** Q: This is the set of nodes of P that are children (in the hierarchy) of some node of Q, at a position considered in s (i.e., the sth children). If $[s]$ is not present, we assume 1..*last*. For example, title **child** chapter retrieves the titles of all chapters (and not titles of sections inside chapters).

—P **parent** (k) Q: This is the set of nodes of P that are parents (in the hierarchy) of at least k nodes of Q. If (k) is not present, we assume 1. For example, chapter **parent** (3) section selects chapters with three or more top-level sections.

—*Distance operators:* These select from the first operand elements that are at a given distance of some element of the second operand, under certain additional conditions.

—P **after/before** Q (C): This is the set of nodes of P whose segments begin/end after/before the end/beginning of a segment in Q. If there is more than one P-candidate for a node of Q, the nearest one to the Q-node is considered (if they are at the same distance, then one of them includes the other, and we select the including one). In order for a P-node to be considered a candidate for a Q-node, the minimal node of C containing them must be the same, or must not exist in both cases. For example, table **after** figure (chapter) retrieves the nearest tables following figures, inside the same chapter.

—P **after/before** (k) Q (C): This is the set of all nodes of P whose segments begin/end after/before the end/beginning of a segment in Q, at a distance of at most k text symbols (not only nearest ones). C plays the same role as above. For example, "computer" **before** (10) "architecture" (paragraph) selects the words "computer" that are followed by the word "architecture" at a distance of at most 10 symbols within the same paragraph. Recall that this distance is measured in the filtered file (i.e., with markup removed).

—*Set manipulation operators:* These manipulate both operands as sets, implementing union, difference, and intersection under different criteria. Except for **same**, both operands must be from the same hierarchy (which must not be the text hierarchy).

—$P + Q$: This is the union of P and Q. For example, small + medium + large is the set of all size-changing commands. To make a union on text segments, use **collapse**.

—$P - Q$: This is the set difference of P and Q. For example, chapter − (chapter **with** figure) are the chapters with no figures. To subtract text segments, we resort to operations on matches.

—P **is** Q: This is the intersection of P and Q. For example, ([1] section **in** chapter) **is** ([3] section **in** page) selects the sections that are first (top-level) sections of a chapter and, at the same time, are the third (top-level) section of a page. To intersect text segments, use **same**.

—P **same** Q: This is the set of nodes of P whose segments are the same segment of a node in Q. P and Q can be from different hierarchies. For example, title **same** "Introduction" gets the titles that say (exactly) "Introduction."

Note that all operations related with beginnings and endings make sense only if the operands are from different hierarchies, since otherwise they are the same as their full-segment counterparts.

3.3.2 *Examples.* We present some examples of the use of the language, to give an idea of what kind of queries can be posed. Suppose we have a hierarchy V with the main structural component book. A book has an introduction, a number of chapters, a bibliography, and an appendix. Each of these may have sections (which may have more sections within them), as well as formulas, figures, and tables. A table is divided into rows, and the rows are divided into columns. A number of paragraphs may appear in chapters, sections, the introduction, and the appendix. The book, chapter, section, figure, and table each always has a title. Finally, we have citations that reference other books, listed in the bibliography.

We have another hierarchy V' with volume, page, and line, and a hierarchy VP for presentation aspects, for example, italics and emphasize. Suppose that we have a simple word-matching language for text:

—chapter **parent** (title **same** "Architecture") is the set of all chapters (of all books) titled "Architecture." Here, "Architecture" is an expression of the pattern-matching sublanguage.

—[last] figure **in** (chapter **with** (section **with** (title **with** "early"))) is the last figure of chapters in which some section (or subsection, **parent** must be used to select top-level sections) has a title that includes the word "early." This query is illustrated in Figure 3.

—paragraph **before** (paragraph **with** ("Computer" **before** (10) "Science" (paragraph))) (page) is the paragraph preceding another paragraph where the word "Computer" appears (at 10 symbols or less) before the word "Science." Both paragraphs must be on the same page.

—[3] column **in** ([2] row **in** (table **with** (title **same** "Results"))) extracts the text in position (2, 3) of tables titled "Results."

—(citations **in** ([2..4] chapter **in** book)) **with** "Knu*" selects references to Knuth's books in Chapters 2–4.

—(section **with** formula) − (section **in** appendix) selects sections with mathematical formulas that are not in appendices.

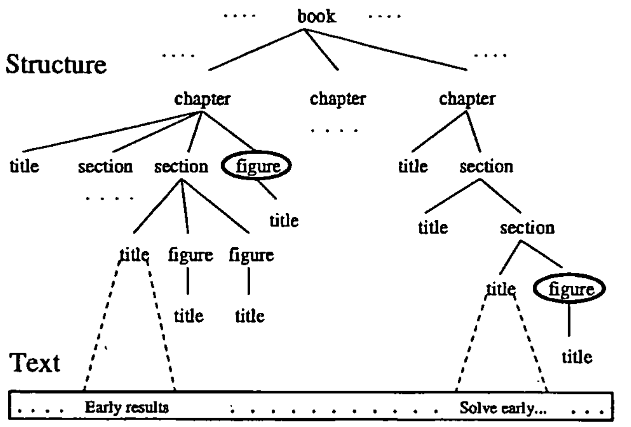

Fig. 3. Illustration of the effect of the query [last] figure **in** (chapter **with** (section **with** (title **with** "early"))). The circles indicate selected nodes.

—introduction + (chapter **parent** (title **with** "Conclusions"))$^-$ + bibliography can be a good abstract of books.

4. EXPRESSIVENESS

We first compare formally our model with the other similar models surveyed (except *p-strings*, since it is a structure manipulation model). Although this point-to-point comparison is useful, no formal categorization of expressiveness features exists. Therefore, we appeal to informal methods. We show that our model fits the quality criteria of Sacks-Davis et al. [1994]. We also develop an informal framework to situate models of this kind.

4.1 A Formal Comparison

In the Appendix we formally define the semantics of our operations. The definition is used to compare our model with each of the novel models, to determine which features from ours can be represented in others and vice versa. A brief abstract of the results follows, omitting the representation mapping performed between the models (see Navarro [1995] for details).

The Hybrid Model [*Baeza-Yates 1996*]. Our model can completely represent it, but the converse is weak. Although it can represent a structure defined in our model, little can be queried about it (e.g., ancestorship).

PAT Expressions [*Salminen and Tompa 1992*]. Our model can almost completely represent it, disregarding some undesirable complications of the language that one really would not want to represent (mainly regarding conversions between regions and match points). The converse is again weak, since this model cannot represent recursive structures, which prevents it from representing almost all of the operations in our language.

Overlapped Lists [*Clarke et al. 1995*]. The comparison is difficult because the models are almost orthogonal. Ours can represent hierarchies but not overlaps, while the overlapped lists model does the inverse. Disregarding the fact that each model cannot represent the most important structure of the other one, most operations can be translated between both models.

Lists of References [*MacLeod 1991*]. Our model can completely represent this model (recall that we consider the part of the model related to querying structures). On the other hand, a good portion of our model can be represented in this one, the most important omissions being the inability to have multiple hierarchies and to return nested components.

Tree Matching [*Kilpeläinen and Mannila 1993*]. We can represent only part of this model. The most important omissions are, of course, logical variables (which make the model implementation NP-complete) and the inclusion semantics. These are different in both models: in tree matching, an inclusion relationship must hold in the text if and only if it holds in the query, whereas in our model, an inclusion relationship must hold in the text if it holds in the query (but more relations can hold in the text). This prevents each model representing the other in this aspect, although we show that we can represent some restricted cases.

This is related to what Consens and Milo [1995] call the *both-included* problem, namely, the ability to express "*a* containing (*b* followed by *c*)." In Consens and Milo [1995], a simplification of PAT expressions is used to formally analyze its expressive power, finding that *both-included* cannot be represented without introducing tuples and join capabilities into the language (á la the relational model). The same holds for our model. Observe that the source of this problem is that it is not possible to express that a name appearing in two parts of an expression should denote the same node, and that is exactly what a relational equijoin would do. By using logical variables, this model can represent almost all of ours, its weak part being the integration of text and structure.

4.2 Quality Criteria

In Sacks-Davis et al. [1994], a number of queries that this kind of language should be able to answer are pointed out. We summarize them here to show that we can express all in the areas we are interested in (i.e., we exclude the features related to relevance ranking and connection to relational databases, which are not addressed in this work):

—Word-by-word access, for example, "find ⟨*doc*⟩s containing 'parallel' and ('computing' or 'processing')" can be expressed as (doc **with** "parallel") **with** ("computing" **collapse** "processing").

—Query scope restricted to subdocuments, for example, "find ⟨*doc*⟩s with ⟨*title*⟩ containing 'parallel' and 'processing'" can be expressed as doc **parent** ((title **with** "parallel") **with** "processing"). The other example in this article is "find ⟨*doc*⟩s with 1st ⟨*para*⟩ containing 'parallel' and 'processing'" which can be expressed as doc **with** ((([1] para **in** doc) **with** "parallel") **with** "processing").

—Retrieval of subdocuments, for example, "find ⟨*section*⟩s with ⟨*para*⟩s containing 'parallel' and 'processing'" can be expressed as section **with** ((para **with** "parallel") **with** "processing").

—Access by structure of documents. Many examples are presented here:
  —"Find elements with parent of type ⟨*article*⟩" can be expressed as All **child** article, where All is the name of the hierarchy.
  —"Find elements with children" can be expressed as All **parent** All.
  —"Find elements where the first child is ⟨*title*⟩" can be expressed as All **parent** ([1] title **child** All).
  —"Find elements within a ⟨*section*⟩" can be expressed as All **in** section.
  —"Find ⟨*doc*⟩s that contain a ⟨*corres*⟩" can be expressed as doc **with** corres.
  —"Find ⟨*section*⟩s that contain a ⟨*section*⟩" can be expressed as section **with** section.

—Access to different types of documents. For example, "Find articles, papers, and books with 'parallel' and 'computing' in the title" can be expressed as (article + paper + book) **with** ((title **with** "parallel") **with** "computer"). This issue is more concerned with the problem of having the different names standing for "title" in each type of document, but this is also easily handled: (book **with** booktitle. . .) + (article **with** articletitle. . .) + ...

—Access by attributes, for example, "find ⟨*corres*⟩s with attribute 'confidential' = yes." If we have those attributes as node-type children of the node and their values in the text, we can answer simple queries; in this case, we express it as corres **parent** (confidential **same** "yes").

4.3 An Informal Framework

We use the experience gained in the formal comparison to define an informal framework to categorize the models according to their important features [Baeza-Yates and Navarro 1996; Navarro and Baeza-Yates 1995a]. This framework divides the analysis into two main areas:

(1) *Structuring power* (i.e., how the database is structured); see Table I.
  —Type of structure (flat, hierarchical, or network)
  —Implicit or explicit structure (i.e., using embedded markup or an explicit structure index)
  —Static or dynamic structure (i.e., ability to reindex)
  —Link between content and structure (strongly text-bound, intermediate, or strongly structure-bound)
  —Structure of answers (flat, overlapped, or nested)

(2) *Query language* (i.e., what can be asked); see Table II.
  —Text matching (i.e., querying content)
  —Set manipulation (i.e., handling sets of answers)

Table I. Analysis of Structuring Power

Model	Type of Structure	Implicit or Explicit	Static or Dynamic	Bound to	Answers
Ours	Hierarchy	Explicit (multiple)	Static	Intermediate	Nested
Hybrid model	Flat	Implicit	Static	Text	Flat
PAT expressions	Hierarchy (not recursive)	Implicit	Dynamic	Text	Flat
Overlapped lists	Hierarchy (with overlaps)	Implicit	Dynamic	Text	Overlapped
Lists of references	Hierarchy (and network)	Explicit (single)	Static	Structure	Flat (and of the same type)
Tree matching	Hierarchy	Explicit (single)	Static	Structure	Nested
p-strings	Hierarchy	Explicit (multiple)	Dynamic	Intermediate	Nested

Table II. Analysis of Query Languages

Model	Set Manipulation	Inclusion Relationships	Distances
Ours	Yes (same hierarchy); a different set for nodes and for text	Including n and included; direct and positional inclusion	Both distance-bound and minimal; inside a given node
Hybrid model	Separate for text and documents; complement	Restricted to fields	Only in matches
PAT expressions	Yes; also, negation of operations	Including n and included	Yes, distance-bound
Overlapped lists	Union and combination	Including and included, plus negations	Combination and "n words"
Lists of references	Yes, but only for nodes of the same type	Including n and included; restricted direct	None
Tree matching	Yes, via logical connectives	Tree patterns + variables	None

—Inclusion relationships (i.e., selecting nodes included in or including others)
—Distances (i.e., selecting nodes at a given distance to others)

There is a third area regarding how the content is seen, but we do not consider it in this work.

Figure 4 presents a simplified graphical version of this comparison. We identify the main points about expressiveness and represent each model as a set containing the aspects it reasonably supports. From this figure, we can see that the main features our model lacks are tuples, semijoin by content (e.g., to retrieve all chapters whose titles appear in this paragraph),

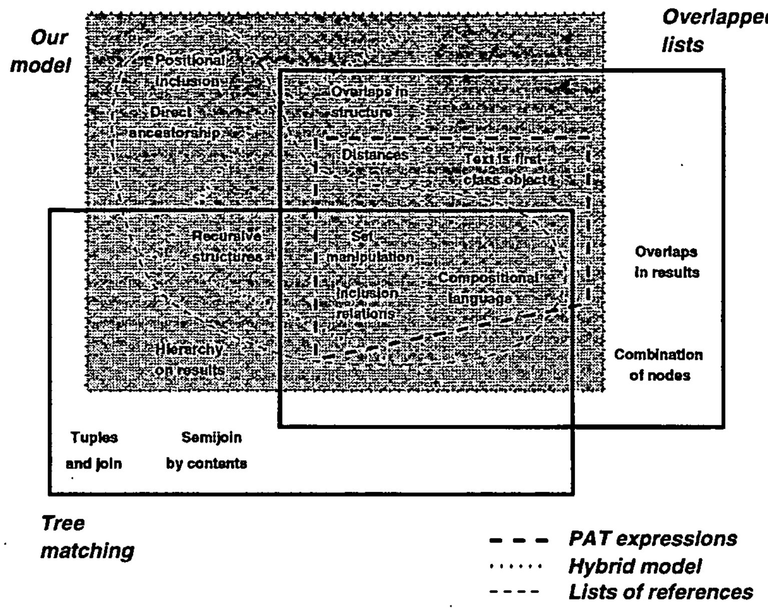

Fig. 4. Graphical representation of the expressiveness comparison.

and the possibility of having overlaps and combining nodes in the result set of a query. We believe that none of them can be included without significantly degrading the performance. The features we support are enough for a large class of applications. Some of the features that are lacking are better included by integrating this model with another one.

5. IMPLEMENTATION ISSUES

In this section we cover the main aspects related to the implementation of the model, in regard to indexing and querying. Later, we depict a suitable software architecture and a prototype we implemented to test the model. More details can be found in Navarro [1995].

We represent each hierarchy as a tree of nodes. These trees form our structure index. From the index, the **Struct** and **Hierarchy** queries obtain other types of trees (rootless) that represent sets of nodes (the algebra on which the rest of the language operates). They also do the processing of content retrieval, by using their own index. The implementation of content retrieval is independent of the rest and is not studied here.

Thus, there are two very different operations: the leaves of the query syntax tree must be solved with an index, while the internal nodes operate and produce sets of nodes. We first explain how to process the internal nodes of the query syntax tree, then the leaves (this includes the indexing scheme), and then the whole evaluation plan. Finally, we formally analyze the efficiency of this implementation.

It is important to observe that we are not proposing any new implementation technique, e.g., those found in Mackie and Zobel [1992]. Rather, we use well-known techniques to implement the model efficiently. The key of the efficiency achieved is in the definition of the model, which allows an efficient implementation with classical tools.

5.1 Evaluating the Internal Query Nodes: Traversal Algorithms

Since the language is compositional, all of the operations except **Struct** and **Hierarchy** receive and deliver sets of nodes. These sets are also arranged into trees, attending to their ancestorship in the corresponding hierarchy (text queries return flat sets, implemented as rootless trees with all of their nodes in the first level).

Since only proximal nodes are related in the operations, all of the algorithms traverse both trees in synchronization. The idea is much like list merging, but in this case each node has a next sibling and a list of children. Each operation traverses the trees in a slightly different way and performs slightly different operations, but the central idea is the same.

Two implementations are possible: a full evaluation scheme computes the entire set of answers at once, while a lazy evaluation scheme computes only the result, and nodes from inner operands of the query syntax tree are obtained only if they are necessary to compute the final result. The lazy mechanism works as follows: from the syntax tree of the query, we require the first level of answers. This triggers new requirements to the children of the root of the syntax tree, which in turn expand the first level of their answers, and so on.

Hopefully, not all of the sets involved in the expression need to be fully evaluated. This mechanism is not new; for example, it is of widespread use in lazy functional languages and in object-oriented query languages.

The lazy version forces an order of evaluation that is not always optimal, since it is given by the requirements of operators higher in the query syntax tree. Because of this, it has higher complexity. Since, on the other hand, it may compute only part of the result, it is not immediate which one is better. Experimental results (as demonstrated later) show that lazy evaluation is normally better. Lazy evaluation is also suitable for interactive environments in which the user wants to see a top-level answer and then navigates only into some subtrees, thus avoiding the need to evaluate the rest.

5.2 Evaluating the Leaves of the Query Tree: Indexing Schemes

What is left to consider is how to efficiently solve **Struct** and **Hierarchy** queries. These are handled by accessing an index. The requirements for this index are as follows:

(1) given a structural component ID, retrieve the tree of all nodes of that type and
(2) given a hierarchy, retrieve the tree of all its nodes.

For lazy evaluation, instead, we keep a pointer to a node in the disk, and ask it to do the following

(3) given a node, retrieve all of its top-level descendants of the same type and
(4) given a node, retrieve all of its children.

These operations must preferably be linear (in the size of the answer), counting both CPU processing, number of disk accesses, and total seek time. Observe that the total seek time may be of higher order than the number of disk accesses; for example, $O(n)$ random accesses to a file take $O(n^2)$ seek time, on average.

There are many alternatives to handling the trees of the structure index on disk. Each one leads to a different indexing scheme.

5.2.1 *A Full Index.* A full index stores the hierarchy tree in a breath-first layout on disk. For each node, enough pointers are kept to perform (1) and (2) in linear expected time with respect to the size of the output (although the seek time for (1) is proportional to the size of the whole hierarchy).

The index is split into one file per level of the tree, to ease the reindexing process. With the same purpose, the endpoints of the segments of each node are computed relative to their parent segment. This does not pose any implementation problem, since a node is only accessed via its parent. The benefit is that complete subtrees can be inserted or deleted without modifying the numbers in the rest of the index. When the results of **Struct** and **Hierarchy** queries are extracted from the index, the positions are rewritten to be absolute.

Although in most cases it suffices to retrieve the node ID and the segment of each node in order to further process the query, sometimes the node ID of the parent is also needed (for **parent** and **child** queries). It is possible to determine syntactically from the query when this is needed, to avoid any further access to the disk. If there is enough memory, it is more efficient to read, in a single pass, all of the **Struct** nodes needed for all of the leaves of the query syntax tree.

Observe that this index is not well suited for lazy evaluation, since it cannot efficiently implement (3). This is because the top-level descendants of a node of the same type can be spread across the file, even unordered, so the seek time may be large. Another drawback of this index is its large space requirement (seven words per node).

5.2.2 *A Partial Index.* It is possible to reduce the space requirement to just two words per node, at the cost of not allowing **parent/child** queries or two different nodes having the same segment (this last restriction is easily overcome at no cost of expressiveness). In this case, two files are kept for each node type, one with the sorted start points and the other with the sorted end points (see Figure 5).

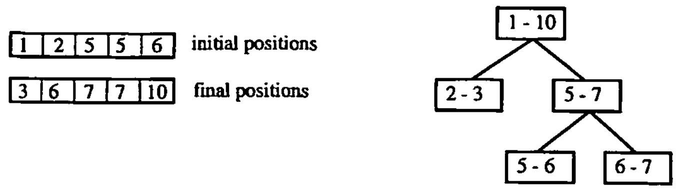

Fig. 5. Example of a partial index and its associated hierarchy.

It is easy to implement requirement (1) in linear time and in a single pass over the index, but for (2) the **Hierarchy** query must be solved as the union of its node types (this takes $\approx kn/2$ time, n being the size of the hierarchy and k being the number of different node types). Finally, the index must be wholly rewritten upon reindexing, and this scheme is worse than the previous one for (3).

5.2.3 *An Index Suitable for Lazy Evaluation.* With three words per node, a version of the partial index suitable for lazy evaluation can be designed. This can be extended to a full index of four words per node (by adding the parent of each node), but it still would not be able to handle different nodes with the same segment.

In this case, a file for each node type is kept, with a layout similar to the full index. This allows the implementation of (1) and (3) in linear time, but (2) and (4) must be solved as before. This is not a big problem with lazy evaluation, since the whole hierarchy is probably never computed. The problem of different nodes having the same segment can be solved by the parser, with artificial intermediate points.

5.3 The Whole Query Evaluation: Query Plan

Given a query syntax tree, any evaluation order that respects the dependencies is allowed. We prefer the one that minimizes the total amount of memory needed for the whole evaluation. The sequence of operations to perform to solve the query is called a *query plan.*

The problem of generating minimum-memory query plans is solved, although simplified, in Aho et al. [1986]. One first has to obtain the larger operand, then obtain the other operand, and then operate them. Since in Aho et al. [1986] the problem is to use registers to compile an expression tree, the weight of a tree is defined as the number of its nodes. We should instead estimate the size of our sets. In absence of good estimators, using the number of nodes seems a reasonable choice. Thus, the algorithm would be as simple as solving the tree by evaluating the bigger subtree first.

A useful alternative to a syntax tree for representing the query is to have a directed acyclic graph (DAG), to avoid reevaluating common subexpressions. In this case, the problem of finding an optimal evaluation order becomes much more complicated, being similar to the problem of evaluating the DAG of an expression that minimizes registers, which is known to be

Table III. Time Complexities of the Algorithms

Operation	Full	Lazy
+, −	n	$n \min(d, h)$
is/same	n	n
in	$\min(n, d^2h)$	$\min(n, d^2h)$
beginin/endin	$\min(n, d^2h)$	$\min(n + dh, d^2h)$
[s]***in**	$n \min(d, h)$	$n \min(d, h)$
with*(k)	$k = 1 \ ? \ n : nh$	$n \min(n, k + dh)$
[s] **child**	n	n
parent(k)	n	$n \min(d, h)$
after/before (C)	$C = \emptyset \ ? \ n : n \min(n, dh)$	$C = \emptyset \ ? \ n : n \min(n, dh)$

n is the size of the operands; h is the maximum height of their tree representation; and d is the maximum arity of those trees.

NP-complete [Garey and Johnson 1979]. Some heuristics for this case are presented by Aho et al. [1986].

These policies for evaluating a query in a given order are only applicable to full evaluation. Lazy evaluation expands the nodes in an unpredictable way, which for each node of the syntax tree is dictated by the requirements posed by its parent.

Another important point is that we can write our algorithms to operate by modifying one of the operands to produce the answer or by generating a new set. If the operand is to be used only once, it is better to modify it; otherwise, we should generate a new set. The query plan generator must implement the appropriate policy to avoid keeping unnecessary copies in memory, deleting or replacing operands the last time they are used. Another interesting point is the optimization of the query, but we do not address this issue here, since it is complex enough to constitute a whole separate problem.

If, despite the clever algorithms to avoid it, the operands are too large to fit into memory, a swapping policy must be implemented. The problem can be solved by a virtual-memory-like approach, keeping part of the intermediate results swapped out to disk. In this case, we must select the operand that will be used later to swap it out (this information is available from the query plan). An interesting option to store those internal results is to use the same layout as the one we use for the indices of node types. This idea together with a good swapping policy provides a uniform and elegant solution to the problem.

5.4 Analysis

We analyzed the efficiency of our algorithms both theoretically and experimentally. Table III abstracts the complexity analysis. Observe that the lazy version, as expected, has higher complexity than the full one. See Navarro [1995] for more algorithmic details. The performance of **Hierarchy** and **Struct** queries depend on the indexing scheme, explained in Section 5.2.

6. A SOFTWARE ARCHITECTURE

In this section we outline a possible software architecture for a system based on our model. Users should interact with the system via an interface, in which they define what they want in a friendly language (e.g., Kuikka and Salminen [1995] and Kilpeläinen and Mannila [1993]). This interface should convert the query into a query syntax tree, i.e., in the language we present here. This tree is then submitted to the query engine.

The query engine optimizes the query and generates a smart query plan to evaluate it (i.e., converts the tree into a sequence of operations to perform). The leaves of the query tree involve extracting components of the hierarchy by name (node types) and text-matching subexpressions. The first ones are solved by accessing the index on the structure to extract the whole set of nodes of that type (i.e., a set of node IDs and their segments). The second ones are submitted to the text search engine, which returns a list of segments corresponding to matched portions of the text. Thereafter, the rest of the operations are performed internally, until the final result (a set of nodes) is delivered back to the interface.

The interface is in charge of visualizing the results. To accomplish this, it must access the contents of the database, at the portions given by the retrieved segments. This is also done via a request to the text engine, since only that engine knows how to access the text.

The text engine is in charge of offering a text pattern-matching language, keeping the indices it needs for searching, and presenting a filtered version of the text file to upper layers. The first service accepts a query and returns a list of matching segments. The third one accepts a segment and retrieves its text contents.

If the text engine is a completely separate subsystem, two separate indexing processes may exist. One of them indexes the text to answer text pattern-matching queries (this indexing is performed by the text engine). The other extracts the structure in some way from the text (parsing, recognizing markup, etc.) and creates the structure index, which is later accessed by the query engine. This is the only time when the text can be accessed directly from outside of the text engine. Indeed, both indexers must collaborate, since the markup used by the structure indexer should be filtered out by the text indexer when presenting the text to upper layers.

See Figures 6 and 7 for diagrams of how a complete system based on this scheme could be organized. The "document layer" is intended to support more sophisticated document management, such as collections of documents.

7. EXPERIMENTAL RESULTS

We implemented a prototype following the proposed software architecture, to test average time and space measures, as well as to evaluate heuristics. Our aim is not to compare our model with other systems, which is very difficult in practice, but to test the suitability of our model for implementation.

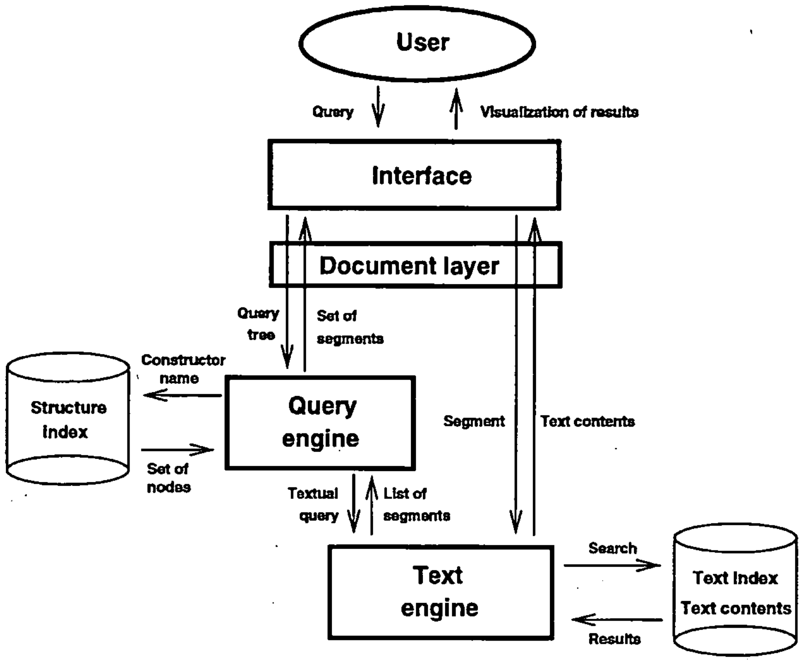

Fig. 6. Architecture of a system following our model, in regard to querying.

For the matching sublanguage, we used the API of SearchCity [Ars Innovandi 1992], which is based on the use of PAT or suffix arrays [Frakes and Baeza-Yates 1992; Manber and Myers 1990]. The matching sublanguage supported by this API includes whole words, ranges, wildcards, proximity search, Boolean operators, subtraction, and fuzzy search. The API allows filtering of the raw text, by applying a format filter, character normalization filter, and a synonyms and stopwords filter. The first filter allows one to use files in other formats different from ASCII, without copying their filtered form into a new file. All format-related (i.e., nonsearchable) portions of the file are filtered out, so that queries can only see the true contents of the file.

We use this filtering facility to incorporate texts whose structure is embedded into their content, in this way allowing only the contents to be searchable and using the marking to parse the structure and to generate a structure index. Hence, we have an index for matches and a separate index per hierarchy. The PAT array can be seen as a generic index for the text hierarchy.

We have implemented filters for DDIF [Digital 1991], SGML [ISO 1986], Latex [Lamport 1986], and C code [Kernighan and Ritchie 1978]. The query plan generation is still very simple; no query optimization is performed.

Currently, the structure index is kept in memory, although in the future it will be kept on disk. Both the text and the text index are kept on disk. This is not as serious a limitation as it appears, since the structure index

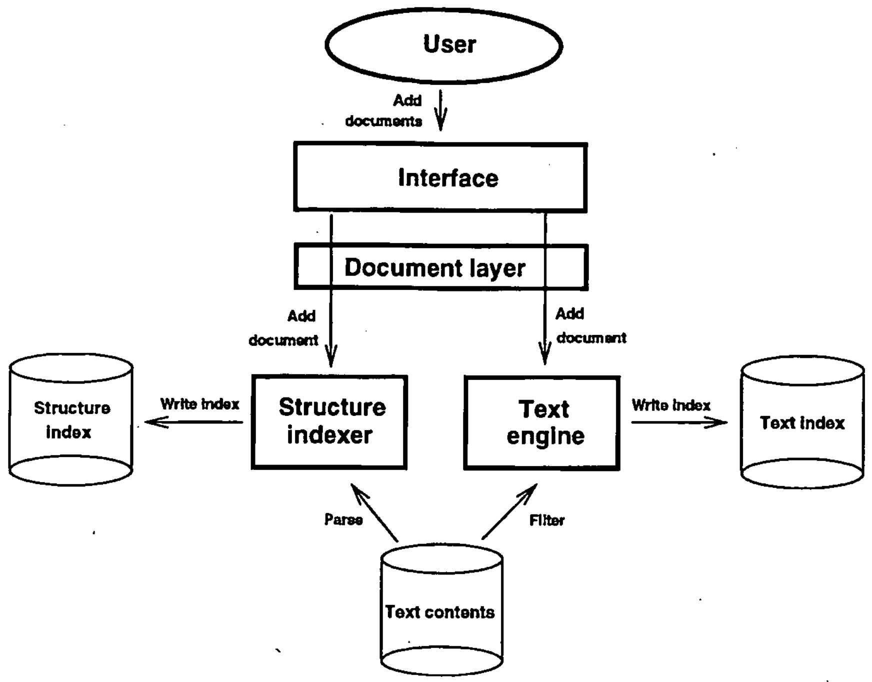

Fig. 7. Architecture of a system following our model, in regard to indexing.

tends to be far smaller than the text. For example, one megabyte of texts and articles generates nearly 1000 nodes, that is, an index of 28 kilobytes using the most space-demanding indexing scheme. With only eight megabytes of RAM devoted to the index, we can handle a database of 300 megabytes of text. There are, of course, other types of documents that pose more serious problems. For example, one megabyte of C code indexed via a fine-grain parsing can generate an index of 300,000 nodes, that is, up to 10 megabytes for the full index.

We conducted a set of tests running our prototype implementation on a Sun SparcClassic, with 16 megabytes of RAM, running SunOS 4.1.3_U1. The CPU speed of this machine is approximately SpecMark 26.

From these results, we conclude that, in the full version, the time to process a query is proportional to the total number of nodes of all internal results, the constant being nearly 50,000 nodes per second on our machine. A rough approximation is $(2q - 1) \times$ (*average operand size*), where q is the number of nodes of the query syntax tree. The lazy version is normally better than the full one, especially for complex queries, although its running times are unstable. The running times are between 25% and 90% of the full version, and between 40% and 100% of the nodes are computed. Figure 8 shows typical times for a single operation.

These good results are possible thanks to our approach for operating proximal nodes, which allows us to compute the results by a one-pass

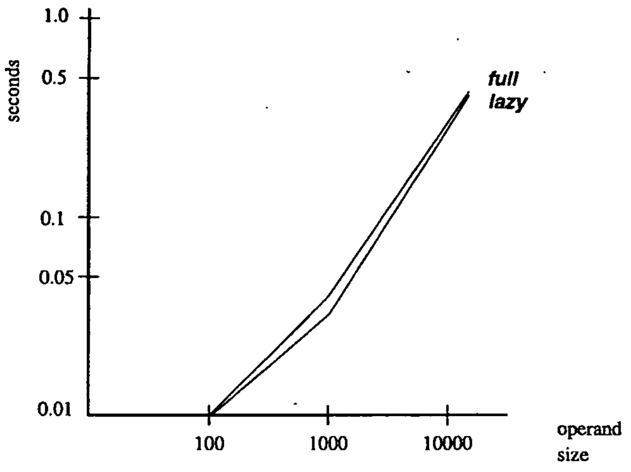

Fig. 8. Typical times for a single operation query. Note that the scale is logarithmic in both axes.

traversal through the operands. The ideas of a set-oriented query language, a data structure to efficiently arrange segments, and the reduction of all queries to operations on proximal nodes lead to an implementation where the amortized cost per processed element is, in most cases, constant.

8. EXTENDING THE MODEL TO MULTIMEDIA DOCUMENTS

In this section we study how we can extend our model to handle other types of data. We are interested in querying documents containing not only text, but also audio, images, video, etc. [Subrahmanian and Jajodia 1996].

Multimedia documents have normally a structure that relates their components, be they text, images, etc. We call this structure the "external structure." However, the text as well as the multimedia objects can have an internal structure. For example, it is possible to structure an audio segment corresponding to a symphony.

Although text has been normally queried by content, in classical multimedia databases multimedia objects were not queried truly by content [Bertino et al. 1988]. Rather, a textual or formatted data description (which is created by hand) was attached to each object, and that was the only information queries could ask on them. Multimedia data were in this way introduced into the classical formatted data model, leaving only performance and security considerations unresolved [Elmasri and Navathe 1994].

Recently, advances in signal processing and artificial intelligence techniques have allowed the possibility of searching by content in objects such as audio and images (e.g., Christodulakis and Faloutsos [1986] and Wu et al. [1994]). Our purpose is to show how these capabilities can be included in a data model to integrate the new facilities smoothly into existing capabilities. Moreover, we show how the structure inherent in multimedia documents, both external to the multimedia objects and internal to each one, can be exploited to improve the retrieval capabilities of a database.

For our fundamental ideas to remain applicable, we assume that it is still possible to impose a hierarchical structuring on the data. This assumption is quite general, although simpler than hypermedia or spatial structuring. Hypermedia structuring could be supported by differentiating between "structure" links (which are hierarchical) and "pointer" links (which are to be queried by other means, e.g., a Graphlog-like language [Consens and Mendelzon 1993]).

For technical reasons (i.e., to represent structure by containment of segments), we assume that the different elements inside each structure can be linearly ordered. This is done internally, and it is *not* visible to the user. Queries about distances are not permitted to spread among different types of objects (we return to this later). If the document does not have order between elements, positional queries can be simply not permitted, hiding the (artificial) ordering from the user.

Thus, our database is now composed of

—*data*, which is a sequence of segments of heterogeneous types, and
—*structure*, as before.

Each relevant position of the data stream is assigned a number. These numbers are in ascending order, but they no longer represent consecutive text symbols. They just indicate the ordering (and containment) between the elements. Because of this, our structural **after/before** operation no longer makes sense and must be replaced by a medium-specific **after/before** where applicable (e.g., positions may represent words in the text and seconds in the audio). Distances make sense only inside a homogeneous data portion or if they contain complete data portions, where the interpretation of their start/end points is specific to that medium.

Instead of a single sublanguage for (text) content, we have such a language for each medium. For example, the language for audio contents may consist of audio pattern matching (by signal processing); the one for images may be oriented toward (precomputed) semantic descriptions. The answers are regarded as segments from a specific hierarchy (one for each medium, and not only the text hierarchy).

Although each medium interprets the structure in its own way, we must keep the consistent idea of containment. For example, it may represent a part-of relation on the objects appearing in an image, each node representing an object with an associated "segment." Other queries (e.g., spatial proximity) are to be regarded as medium-specific.

This technique allows us to consider the structure of a document as homogeneous, regardless of whether it is external or internal. All of the queries about, for example, containment, performed on the structure extend seamlessly to the structure defined inside images, audio, etc.

From the software point of view, there is a module responsible for handling each type of data. A general parser must recognize the external

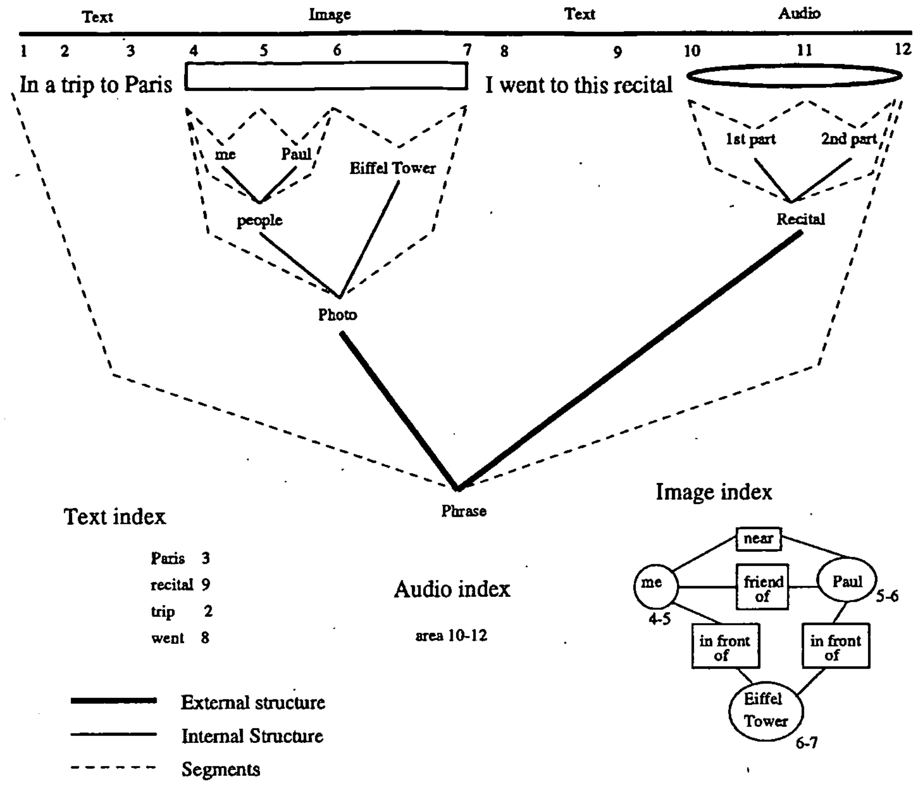

Fig. 9. Document with multimedia elements, with its structure and content indices.

structure and each data object, passing the relevant segment to the corresponding indexer, who indexes its content and internal structure. An index for each medium is built with the content, while the internal structure is returned to the general indexer, which appends it to the hierarchy. At the moment of querying, subexpressions belonging to the content-retrieval sublanguage of each medium are assigned to a specific query processor, which uses its own index (for text retrieval, audio matching, image semantics, etc.) to retrieve a set of segments that represents the relevant portion of the object. The structural part of the query is solved as before, regardless of its external or internal nature.

To visualize the final result, the interface sends the segment to a module that determines which medium it belongs to and passes the request to the appropriate specific module. This may result in the retrieval of a text portion, an image, a combination of text and audio, etc.

Figure 9 shows an example document. A query such as Recital **in** (Phrase **with** (Photo **with** (Paul near me))) retrieves the segment 10–12, visualized as the sound recording of the recital.

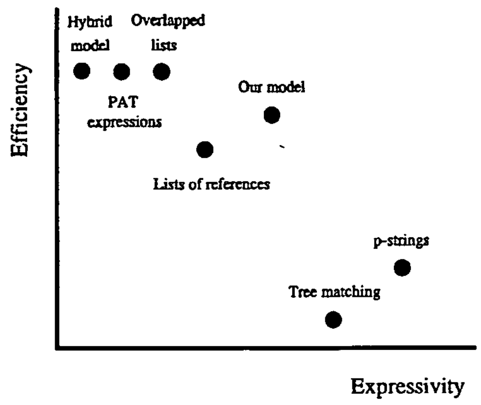

Fig. 10. Comparison between similar models, in regard to both efficiency and expressiveness.

9. CONCLUSIONS AND FUTURE WORK

The problem of querying a document database on both its content and structure has been analyzed. We found the existing approaches to be either not expressive enough or inefficient.

Hence, we have defined a model for structuring and querying textual databases that is expressive enough and efficiently implementable and have extended it to handle other types of data. The language is not meant to be accessed by final users, but to constitute the operational algebra.

We have evaluated our model in expressiveness and efficiency. We showed it to be competitive in expressiveness, getting close to others that do not have an efficient implementation. On the other hand, the algorithms show good performance, both in their analysis and in the experimental tests. This situates our model close in efficiency to those that are much less expressive.

See Figure 10 for a graphical (and informal) comparison of similar models taking into account both efficiency and expressiveness. Note that we have included *p-strings* in this drawing, assuming an expressiveness superior to all of the languages we have analyzed. Note also that only a part of the lists of references model is considered (and the efficiency to implement only that part is considered). Observe that, as with any quantization of concepts, this comparison is subjective. Nevertheless, it does give an idea of where our model lies.

There are a number of research directions related to this work:

—Further exploration of the possibilities offered by the model in order to find more operators that fit into our philosophy (being thus efficiently implementable).

—Definition of a query language suitable for end users, possibly visual, to map onto our operational algebra (e.g., see Kuikka and Salminen [1995]).

—Integration of this kind of model and others, such as the relational model or the traditional ones of information retrieval. This issue has not been considered here, since we focus on the structure problem. An important issue is how to include relevance ranking in our model (see Sacks-Davis et al. [1994] and Arnold-Moore et al. [1995] for some ideas).

—Generalization of the problem to manage nonhierarchical structures, such as a hypertext network, while keeping the desirable properties obtained for this simpler case. A recent work [Dao et al. 1996] that extends overlapped lists to handle nesting and overlapping simultaneously shows a different trend that deserves attention.

—A formal framework in which to compare expressiveness. The long-term goal is a formal and sound hierarchy like what can be found in the area of formal languages (see Navarro and Baeza-Yates [1995a] and Consens and Milo [1995] for some examples).

—A number of implementation issues, for example, designing a good swapping policy for query evaluation, keeping the index on disk, handling multimedia documents, query optimization, etc.

APPENDIX

A. FORMAL SYNTAX AND SEMANTICS

We first define our formal model, then the syntax of the expressions we have studied (*Expr*) by an annotated abstract syntax, and finally the semantics of those operations by a function $\mathcal{S} : Expr \rightarrow \wp(\mathcal{N})$ (i.e., from expressions to sets of nodes).

A.1 Formal Model

A text database is a tuple $(\mathcal{T}, \mathcal{V}, C, N, R, Type, Segm)$, where

—$\mathcal{T} : [1..T] \rightarrow \Sigma$ is the text array. T is the size of the database (number of symbols), and Σ is the alphabet of the text.

—$\mathcal{V}$ is the finite set of hierarchies over the text, with a distinguished element $V_t \in \mathcal{V}$ (the text hierarchy).

—$C : \mathcal{V} \rightarrow \wp(\mathcal{C})$ is the set of node types of each hierarchy; we also write $C(V)$ as C_V. $\mathcal{C}$ is the finite set of node types, with a distinguished element $C_t \in \mathcal{C}$ (the text node type). $\forall V_1 \neq V_2 \in \mathcal{V}$, $C_{V_1} \cap C_{V_2} = \emptyset$ holds. Also, $C_{V_t} = \{C_t\}$.

—$N : \mathcal{V} \rightarrow \wp(\mathcal{N})$ is the set of nodes of each hierarchy; we also write $N(V)$ as N_V. $\mathcal{N}$ is the finite set of nodes, including special text nodes $t_{a,b}$ for each $1 \leq a \leq b \leq T$ (the text nodes). $\forall V_1 \neq V_2 \in \mathcal{V}$, $N_{V_1} \cap N_{V_2} = \emptyset$ holds. Also, $N_{V_t} = \{t_{a,b}/1 \leq a \leq b \leq T\}$.

—$R : \mathcal{V} \rightarrow \wp(\mathcal{N} \times \mathcal{N})$ is the binary relationship that defines the tree of each hierarchy; we also write $R(V)$ as R_V. $\forall V \in \mathcal{V}$, $R_V \subseteq (N_V \times N_V)$ holds. Also, $R(V_t) = \emptyset$.

—$Type : \mathcal{N} \rightarrow \mathcal{C}$ is the type of each node. $\forall V \in \mathcal{V}$, $\forall x \in N_V$, $Type(x) \in C_V$ holds. This implies that $\forall a, b/1 \leq a \leq b \leq T$, $Type\ (t_{a,b}) = C_t$.

—$Segm : \mathcal{N} \rightarrow [1..T] \times [1..T]$ is the segment of each node. $\forall x \in \mathcal{N}$, $Segm(x) = (a, b) \Rightarrow a \leq b$ holds. We also define *From* and *To* to satisfy $Segm(x) = (From(x), To(x))$. Finally, we define $Segm(t_{a,b}) = (a, b)$, as expected.

We define a binary relationship $\rightarrow$ as the union of R_V, for all $V \in \mathcal{V}$; that is, $\rightarrow = \cup_{V \in \mathcal{V}} R_V$. We impose the following conditions on $\rightarrow$:

—$\forall x, y \in \mathcal{N}, x \rightarrow^+ y \Rightarrow \neg\, y \rightarrow x$; that is, loops are not allowed. Here, $\rightarrow^+$ is the transitive closure of $\rightarrow$.

—$\forall V \in \mathcal{V} - \{V_t\}, \exists!\ r_V \in N_V / \nexists x \in N_V / x \rightarrow r_V$; that is, each hierarchy except the text hierarchy has a single root.

—$\forall x, y \in \mathcal{N}, x \rightarrow y \Rightarrow \nexists z \neq x / z \rightarrow y$; that is, any node has at most one parent.

—$\forall x, y \in \mathcal{N}, x \rightarrow y \Rightarrow Segm(y) \subseteq Segm(x)$. When we operate segments as sets, we interpret $Segm(x) = \{n \in Nat / From(x) \leq n \leq To(x)\}$. That is, the segment of a node includes the segment of its descendants. *Nat* is the set of natural numbers.

—$\forall V \in \mathcal{V} - \{V_t\}, \forall x, y \in N_V, Segm(x) \subset Segm(y) \Rightarrow x \rightarrow^+ y$; that is, except in the text hierarchy, if two segments of the same tree are included one into the other, then the including one is the ancestor of the included.

—$\forall V \in \mathcal{V} - \{V_t\}, \forall x, y \in N_V, Segm(x) = Segm(y) \Rightarrow x \rightarrow^* y \lor y \rightarrow^* x$; that is, except in the text hierarchy, if two segments of the same tree are equal, then they are in a single path of the tree. Here, $\rightarrow^*$ is the Kleene (transitive and reflexive) closure of $\rightarrow$.

—$\forall V \in \mathcal{V} - \{V_t\}, \forall x, y \in N_V, Segm(x) \subseteq Segm(y) \lor Segm(y) \subseteq Segm(x) \lor Segm(x) \cap Segm(y) = \emptyset$; that is, there is a strict hierarchy of segments (except in the text hierarchy).

Finally, we define a binary relation in $\mathcal{N} \times \mathcal{N}$, called $\subset$, to mean that the first node includes the other (do not confuse this with segment inclusion). If both nodes are from the same hierarchy, then the second node must descend from the first one; otherwise, we test for segment inclusion. Thus $x \subset y \Leftrightarrow (\exists V \in \mathcal{V} - \{V_t\} / \{x, y\} \subseteq N_V)\ ?\ y \rightarrow^+ x : Segm(x) \subseteq Segm(y)$. Observe that $x \subset y \Rightarrow Segm(x) \subseteq Segm(y)$, but that the reciprocal is not true.

A.2 Abstract Syntax and Formal Semantics

Figure 11 shows the abstract syntax of the language. In the figure we use *Nat* as the set of natural numbers, *Int* as the integers, *Mat* as the set of pattern-matching expressions, *Pos* as the language for denoting positions, and $E, E_1, E_2, E_3 \in Expr$.

Some compositions are not allowed when the operands are from different hierarchies. We indicate at the right side of each alternative, between brackets, the conditions on the intervening hierarchies for that production to be valid. The hierarchy to which the result belongs is expressed as a

$$\begin{array}{lll}
\mathit{Expr} & \longrightarrow & \textbf{Hierarchy}(V)\ [V \in \mathcal{V} - \{V_t\}, \tau = V] \\
 & | & \textbf{Struct}(c)\ [c \in \mathcal{C} - \{C_t\}, \tau = V / c \in C_V] \\
 & | & \textbf{Match}(m)\ [m \in \mathit{Mat}, \tau = V_t] \\
 & | & (E_1\ \textbf{collapse}\ E_2)\ [\tau = \tau(E_1) = \tau(E_2) = V_t] \\
 & | & \text{... (other operations to manipulate matches)} \\
 & | & (E_1\ +\ E_2)\ [\tau = \tau(E_1) = \tau(E_2) \neq V_t] \\
 & | & (E_1\ -\ E_2)\ [\tau = \tau(E_1) = \tau(E_2) \neq V_t] \\
 & | & (E_1\ \textbf{is}\ E_2)\ [\tau = \tau(E_1) = \tau(E_2) \neq V_t\ ] \\
 & | & (E_1\ \textbf{same}\ E_2)\ [\tau = \tau(E_1)] \\
 & | & (E_1\ \textbf{with}(k)\ E_2)\ [k \in \mathit{Nat}, \tau = \tau(E_1)] \\
 & | & (E_1\ \textbf{withbegin/withend}(k)\ E_2)\ [k \in \mathit{Nat}, \tau = \tau(E_1) \neq \tau(E_2)] \\
 & | & (E_1\ \textbf{in}\ E_2)\ [\tau = \tau(E_1)] \\
 & | & (E_1\ \textbf{beginin/endin}\ E_2)\ [\tau = \tau(E_1) \neq \tau(E_2)] \\
 & | & ([s]\ E_1\ \textbf{in}\ E_2)\ [s \in \mathit{Pos}, \tau = \tau(E_1)] \\
 & | & ([s]\ E_1\ \textbf{beginin/endin}\ E_2)\ [s \in \mathit{Pos}, \tau = \tau(E_1) \neq \tau(E_2)] \\
 & | & (E_1\ \textbf{parent}(k)\ E_2)\ [k \in \mathit{Nat}, \tau = \tau(E_1) = \tau(E_2) \neq V_t] \\
 & | & ([s]\ E_1\ \textbf{child}\ E_2)\ [s \in \mathit{Pos}, \tau = \tau(E_1) = \tau(E_2) \neq V_t] \\
 & | & (E_1\ \textbf{after/before}\ E_2\ (E_3))\ [\tau = \tau(E_1)] \\
 & | & (E_1\ \textbf{after/before}(k)\ E_2\ (E_3))\ [k \in \mathit{Nat}, \tau = \tau(E_1)]
\end{array}$$

Fig. 11. Abstract syntax.

function $\tau : \mathit{Expr} \rightarrow \mathcal{V}$. Between the brackets we also indicate as simply τ the type to which the result of the production belongs.

We are now in the position to define the semantics of the defined operations. We do so by defining a function $\mathcal{I} : \mathit{Expr} \rightarrow \wp(\mathcal{N})$, which interprets each expression in terms of a set of nodes. The function $\mathcal{I}$ is defined inductively as follows:

—$\mathcal{I}(\textbf{Hierarchy}(V)) = N_V$.

—$\mathcal{I}(\textbf{Struct}(c)) = \{x \in \mathcal{N} / \mathit{Type}(x) = c\}$.

—Let m be a pattern-matching expression, whose result is a set of segments $(a_1, b_1)\ldots(a_k, b_k)$. Then, $\mathcal{I}(m) = \{t_{a_i,b_i} / i \in [1..k]\}$.

—$\mathcal{I}$ $(P\ \textbf{collapse}\ Q) = \{t_{a_1,b_n} / \exists t_{a_1,b_1} \ldots t_{a_n,b_n} \in \mathcal{I}(P) \cup \mathcal{I}(Q) / (\forall i,\ b_i \leq a_{i+1}) \wedge \nexists t_{x,y} \in \mathcal{I}(P) \cup \mathcal{I}(Q) - \{t_{a_1,b_1} \ldots t_{a_n,b_n}\} / (x, y) \cap (a_1, b_n) \neq \emptyset\}$.

—$\mathcal{I}(P + Q) = \mathcal{I}(P) \cup \mathcal{I}(Q)$.

—$\mathcal{I}(P - Q) = \mathcal{I}(P) - \mathcal{I}(Q)$.

—$\mathcal{I}(P\ \textbf{is}\ Q) = \mathcal{I}(P) \cap \mathcal{I}(Q)$.

—$\mathcal{I}(P\ \textbf{same}\ Q) = \{x \in \mathcal{I}(P) / \exists y \in \mathcal{I}(Q) / \mathit{Segm}(x) = \mathit{Segm}(y)\}$.

—$\mathcal{I}(P\ \textbf{with}\ (k)\ Q) = \{x \in \mathcal{I}(P) / |\{y \in \mathcal{I}(Q) / y \subset x\}| \geq k\}$.

—$\mathcal{I}(P\ \textbf{withbegin}(k)\ Q) = \{x \in \mathcal{I}(P) / |\{y \in \mathcal{I}(Q) / \mathit{From}(y) \in \mathit{Segm}(x)\}| \geq k\}$.

—$\mathcal{I}(P\ \textbf{withend}(k)\ Q) = \{x \in \mathcal{I}(P) / |\{y \in \mathcal{I}(Q) / \mathit{To}(y) \in \mathit{Segm}(x)\}| \geq k\}$.

—$\mathcal{I}(P\ \textbf{in}\ Q) = \{x \in \mathcal{I}(P) / \exists y \in \mathcal{I}(Q) / x \subset y\}$.

—$\mathcal{I}(P\ \textbf{beginin}\ Q) = \{x \in \mathcal{I}(P) / \exists y \in \mathcal{I}(Q) / \mathit{From}(x) \in \mathit{Segm}(y)\}$.

—$\mathcal{I}(P\ \textbf{endin}\ Q) = \{x \in \mathcal{I}(P) / \exists y \in \mathcal{I}(Q) / \mathit{To}(x) \in \mathit{Segm}(y)\}$.

—$\mathcal{I}([s]$ P **in** $Q) = \cup_{y \in \mathcal{I}(Q)} \{x \in \mathcal{X}_y / \mathcal{S}(s, x, \mathcal{X}_y)\}$. Here, $\mathcal{S} : \mathrm{S} \times \mathcal{N} \times \wp(\mathcal{N}) \rightarrow \{true, false\}$ is the interpretation of the position language S, which says whether the left-to-right position of a node in a set of nodes is acceptable by the specification of s. This position is only well defined when none of the segments includes another, which is the case in $\mathcal{X}_y$, which we define as $\mathcal{X}_y = \{x \in \mathcal{I}(P) / x \subset y \wedge x \in maxim(z \in \mathcal{I}(P) / z \subset y \vee y \not\subset z)\}$. *maxim* selects the maximal nodes of a set; that is, $maxim(X) = \{x \in X / \exists y \in X / x \subset y\}$.

—$\mathcal{I}([s]$ P **beginin** $Q) = \cup_{y \in \mathcal{I}(Q)} \{x \in \mathcal{X}_y / \mathcal{S}(s, x, \mathcal{X}_y)\}$. Here, $\mathcal{X}_y = \{x \in \mathcal{I}(P) / From(x) \in Segm(y) \wedge x \in maxim(z \in \mathcal{I}(P) / Segm(z) \not\supset Segm(y))\}$.

—$\mathcal{I}([s]$ P **endin** $Q) = \cup_{y \in \mathcal{I}(Q)} \{x \in \mathcal{X}_y / \mathcal{S}(s, x, \mathcal{X}_y)\}$. Here, $\mathcal{X}_y = \{x \in \mathcal{I}(P) / To(x) \in Segm(y) \wedge x \in maxim(z \in \mathcal{I}(P) / Segm(z) \not\supset Segm(y))\}$.

—$\mathcal{I}(P$ **parent**(k) $Q) = \{x \in \mathcal{I}(P) / |\{y \in \mathcal{I}(Q) / x \rightarrow y\}| \geq k\}$.

—$\mathcal{I}([s]$ P **child** $Q) = \{x \in \mathcal{I}(P) / \exists y \in \mathcal{I}(Q) / y \rightarrow x \wedge \mathcal{S}(s, x, \{z \in \mathcal{N} / y \rightarrow z\})\}$.

—$\mathcal{I}(P$ **after**(k) Q $(C)) = \{x \in \mathcal{I}(P) / \exists y \in \mathcal{I}(Q) / 0 < From(x) - To(y) \leq k \wedge minim(\{z \in \mathcal{I}(C) / x \subset z\}) = minim(\{z \in \mathcal{I}(C) / y \subset z\})\}$.

—$\mathcal{I}(P$ **after** Q $(C)) = \cup_{y \in \mathcal{I}(Q)}$ $first(\{x \in \mathcal{I}(P) / From(x) > To(y) \wedge minim(\{z \in \mathcal{I}(C) / x \subset z\}) = minim(\{z \in \mathcal{I}(C) / y \subset z\})\})$. Here, $first : \wp(\mathcal{N}) \rightarrow \mathcal{N}$ selects the node in the set with the lowest value of *From*, and if there are more than one, the maximal value. If all of the nodes are from the same hierarchy, this criterion gives exactly one node.

—$\mathcal{I}(P$ **before**(k) Q $(C)) = \{x \in \mathcal{I}(P) / \exists y \in \mathcal{I}(Q) / 0 < From(y) - To(x) \leq k \wedge minim(\{z \in \mathcal{I}(C) / x \subset z\}) = minim(\{z \in \mathcal{I}(C) / y \subset z\})\}$.

—$\mathcal{I}(P$ **before** Q $(C)) = \cup_{y \in \mathcal{I}(Q)}$ $last(\{x \in \mathcal{I}(P) / From(y) > To(x) \wedge minim(\{z \in \mathcal{I}(C) / x \subset z\}) = minim(\{z \in \mathcal{I}(C) / y \subset z\})\})$. *last* is analogous to *first*, selecting the highest value of *To*, or the maximal if they are the same.

ACKNOWLEDGMENTS

We thank Patricio Poblete, Nivio Zivani, Mariano Consens, José Blakeley, Omar Alonso, and the unknown referees for their fruitful comments and careful reading that improved this article.

REFERENCES

AHO, A., SETHI, R., AND ULLMAN, J. 1986. *Compilers: Principles, Techniques and Tools.* Addison-Wesley, Reading, Mass.

ARNOLD-MOORE, T., FULLER, M., LOWE, B., THOM, J., AND WILKINSON, R. 1995. The ELF data model and SGQL query language for structured document databases. In *Proceedings of the 6th Australasian Database Conference.* 17–26.

ARS INNOVANDI. 1992. *Search City 1.1 Text Retrieval for Windows Power Users.* Ars Innovandi, Santiago, Chile.

BAEZA-YATES, R. 1994. An hybrid query model for full text retrieval systems. Tech. Rep. DCC-1994-2, Dept. of Computer Science, Univ. of Chile, Santiago, Chile.

BAEZA-YATES, R. 1996. An extended model for full-text databases. *J. Braz. CS Soc. 3,* 2 (Apr.), 57–64.

BAEZA-YATES, R., AND NAVARRO, G. 1996. Integrating contents and structure in text retrieval. *ACM SIGMOD Rec. 25,* 1 (Mar.), 67–79.

BERTINO, E., RABITTI, F., AND GIBBS, S. 1988. Query processing in a multimedia document system. *ACM Trans. Inf. Syst. 6,* 1 (Jan.), 1–41.

BLAKE, G., BRAY, T., AND TOMPA, F. 1992. Shortening the OED: Experience with a grammar-defined database. *ACM Trans. Inf. System. 10,* 3 (July), 213–232.

CATTELL, R. 1991. *Object Data Management.* Addison-Wesley, Reading, Mass.

CHRISTODULAKIS, S. AND FALOUTSOS, C. 1986. Design and performance considerations for an optical disk-based multimedia object server. *IEEE Comput. 19,* 12 (Dec.), 45–56.

CHRISTOPHIDES, V., ABITEBOUL, S., CLUET, S., AND SCHOLL, M. 1994. From structured documents to novel query facilities. In *Proceedings of ACM SIGMOD 94.* ACM, New York, 313–324.

CLARKE, C., CORMACK, G., AND BURKOWSKI, F. 1995. An algebra for structured text search and a framework for its implementation. *Comput. J.*

CODD, E. 1983. A relational model of data for large shared data banks. *Commun. ACM. 26,* 1 (Jan.), 64–69.

CONKLIN, J. 1987. Hypertext: An introduction and survey. *IEEE Comput. 20,* 9 (Sept.), 17–41.

CONSENS, M. AND MENDELZON, A. 1993. Hy$^+$: A hygraph-based query and visualization system. In *Proceedings of ACM SIGMOD 93.* ACM, New York, 511–516.

CONSENS, M. AND MILO, T. 1994. Optimizing queries on files. In *Proceedings of the ACM SIGMOD 94.* ACM, New York, 301–312.

CONSENS, M. AND MILO, T. 1995. Algebras for querying text regions. In *Proceedings of PODS 95.* ACM, New York, 11–22.

DAO, T., SACKS-DAVIS, R., AND THOM, J. 1996. Indexing structured text for queries on containment relationships. In *Proceedings of the 7th Australasian Database Conference.*

DATE, C. 1995. *An Introduction to Database Systems.* 6th ed. Addison-Wesley, Reading, Mass.

DESAI, B., GOYAL, P., AND SADRI, S. 1986. A data model for use with formatted and textual data. *J. ASIS 37,* 3, 158–165.

DIGITAL. 1991. *CDA-DDIF Technical Specification.* Digital Equipment Corp., Maynard, Mass.

ELMASRI, R. AND NAVATHE, S. 1994. *Fundamentals of Database Systems.* 2nd ed. Benjamin Cummings, Menlo Park, Calif.

FAWCETT, H. 1989. *PAT 3.3 User's Guide.* UW Centre for the New OED and Text Research, Univ. of Waterloo, Ontario, Canada.

FRAKES, W. AND BAEZA-YATES, R. Eds. 1992. *Information Retrieval: Data Structures and Algorithms.* Prentice-Hall, Englewood Cliffs, N.J.

GAREY, M. AND JOHNSON, D. 1979. *Computers and Intractability.* W. Freeman and Company.

GONNET, G. AND TOMPA, F. 1987. Mind your grammar: A new approach to modeling text. In *Proceedings of VLDB 87.* VLDB Endowment Press, Saratoga, Calif., 339–346.

HULL, R. AND KING, R. 1987. Semantic database modeling: Survey, applications and research issues. *ACM Comput. Surv. 19,* 3, 201–260.

ISO. 1986. Information processing—Text and office systems—Standard generalized markup language (SGML). ISO 8879-1986, International Standards Organization, Geneva, Switzerland.

ISO. 1991. Information processing—Text composition—Standard page description language (SPDL). ISO/IEC DIS 10180, International Standards Organization, Geneva, Switzerland.

ISO. 1992. Information technology—hypermedia/time-based structuring language (HyTime). ISO/IEC 10744, International Standards Organization, Geneva, Switzerland.

ISO. 1994. Information technology—Text and office systems—Document style semantics and specification language (DSSSL). ISO/IEC DIS 10179.2, International Standards Organization, Geneva, Switzerland.

KERNIGHAN, B. AND RITCHIE, D. 1978. *The C Programming Language*. Prentice-Hall, Englewood Cliffs, N.J.

KILPELÄINEN, P. AND MANNILA, H. 1992. Grammatical tree matching. In *Proceedings of CPM 92*. 162–174.

KILPELÄINEN, P. AND MANNILA, H. 1993. Retrieval from hierarchical texts by partial patterns. In *Proceedings of ACM SIGR 93*. ACM, New York, 214–222.

KIM, W. AND LOCHOVSKI, F., Eds. 1989. *Object-Oriented Concepts, Databases and Applications*. Addison-Wesley, Reading, Mass.

KUIKKA, E. AND SALMINEN, A. 1995. Two-dimensional filters for structured text. *Inf. Process. Manag.*

LAMPORT, L. 1986. *Latex: A Document Preparation System*. Addison-Wesley, Reading, Mass.

MACKIE, E. AND ZOBEL, J. 1992. Retrieval of tree-structured data from disc. In *Proceedings of the 3rd Australasian Database Conference*. 209–216.

MACLEOD, I. 1991. A query language for retrieving information from hierarchic text structures. *Comput. J. 34*, 3, 254–264.

MANBER, U. AND MYERS, G. 1990. Suffix arrays: A new method for on-line string searches. In *Proceedings of ACM-SIAM 90*. ACM, New York, 319–327.

NAVARRO, G. 1995. A language for queries on structure and contents of textual databases. Master's thesis, Dept. of Computer Science, Univ. of Chile, Santiago, Chile.

NAVARRO, G. AND BAEZA-YATES, R. 1995a. Expressive power of a new model for structured text databases. In *Proceedings of PANEL 95*. 1151–1162.

NAVARRO, G. AND BAEZA-YATES, R. 1995b. A language for queries on structure and contents of textual databases. In *Proceedings of ACM SIGIR 95*. ACM, New York, 93–101.

SACKS-DAVIS, R., ARNOLD-MOORE, T., AND ZOBEL, J. 1994. Database systems for structured documents. In *Proceedings of ADTI 94*. 272–283.

SACKS-DAVIS, R., ZOBEL, J., AND RAMAMOHANARAO, K. 1992. Advanced database systems for text retrieval. In *Proceedings of the 3rd Australian Database Conference*. 1–8.

SALMINEN, A. AND TOMPA, F. 1992. PAT expressions: An algebra for text search. In *COMPLEX 92*. 309–332.

SALTON, G. AND MCGILL, M. 1983. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York.

STONEBRAKER, M., STETTNER, H., LYNN, N., KALASH, J., AND GUTTMAN, A. 1983. Document processing in a relational database system. *ACM Trans. Inf. Syst. 1*, 2 (Apr.), 143–158.

SUBRAHMANIAN, M. AND JAJODIA, S. Eds. 1996. *Multimedia Database Systems*. Springer-Verlag, New York.

TAGUE, J., SALMINEN, A., AND MCCLELLAN, C. 1991. Complete formal model for information retrieval systems. In *Proceedings of ACM SIGIR 91*. ACM, New York, 14–20.

WU, J., ANG, Y., LAM, P., LOH, H., AND DESAI, A. 1994. Inference and retrieval of facial images. *Multimedia Syst. 2*.

Received December 1995; revised December 1996; accepted December 1996

Grammatical Tree Matching*

Pekka Kilpeläinen and Heikki Mannila

University of Helsinki, Department of Computer Science
Teollisuuskatu 23, SF-00510 Helsinki, Finland
e-mail: kilpelai@cs.helsinki.fi, mannila@cs.helsinki.fi

Abstract. In structured text databases documents are represented as parse trees, and different tree matching notions can be used as primitives for query languages. Two useful notions of tree matching, *tree inclusion* and *tree pattern matching* both seem to require superlinear time. In this paper we give a general sufficient condition for a tree matching problem to be solvable in linear time, and apply it to tree pattern matching and tree inclusion. The application is based on the notion of a *nonperiodic* parse tree. We argue that most text documents can be modeled in a natural way using grammars yielding nonperiodic parse trees. We show how the knowledge that the target tree is nonperiodic can be used to obtain linear time algorithms for the tree matching problems. We also discuss the preprocessing of patterns for grammatical tree matching.

1 Introduction

Context-free grammars are often used for describing the structure of text documents [7, 2, 3, 6, 19, 11]. When the structure of the text is described using a grammar, the text itself is represented as a parse tree.

To extract information from a parse tree one needs query language primitives that are oriented towards tree-structured data. *Tree pattern matching* is a widely studied problem with many applications. (See, e.g., [9] and its references.) Consider two ordered and labeled trees P (the *pattern*) and T (the *target*). The trees that can be obtained by attaching new subtrees to the leaves of P are *instances* of P. The task in tree pattern matching is to locate subtrees of the target that are instances of the pattern. (See Fig. 1.)

Denote the number of nodes in pattern P by $|P| = m$ and the number of nodes in target T by $|T| = n$. The tree pattern matching problem can be solved by a trivial algorithm that traverses the target and compares the pattern against each subtree in turn. The $O(mn)$ worst case complexity of the trivial algorithm has only recently been improved by Kosaraju [15] and Dubiner, Galil, and Magen [4]; the algorithm of [4] requires time $O(n\sqrt{m}\, polylog(m))$.

To locate a subtree of the target using tree pattern matching one has to know the exact structure of the relevant parts of the target. For example, in order to locate a subtree where a node v is a descendant of a node u, the pattern must describe the exact path between u and v.

* Work supported by the Academy of Finland.

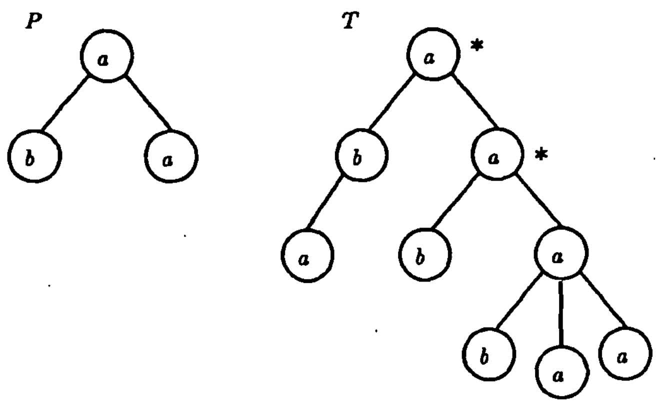

Fig. 1. The subtrees of T that are instances of P are denoted by stars.

In text databases the exact structure of the tree representing the text is often unknown. Thus one needs a more relaxed way of defining the occurrences of the target. Such a way is obtained by choosing the pattern P to stand for trees that *include* P. Intuitively, a tree includes P if a subset of its nodes agrees with the nodes of P with regard to labeling, ancestorship and left-to-right ordering. The precise definition of tree inclusion is given later; for an example see Fig. 2. Note that

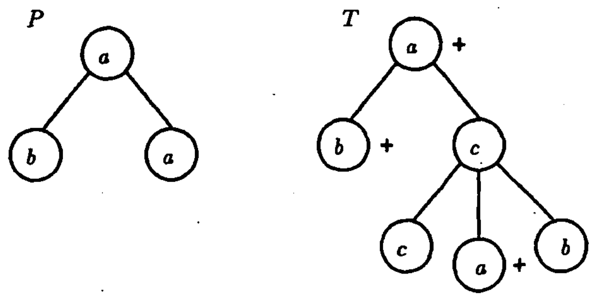

Fig. 2. A pattern P and an including tree T. The nodes of T corresponding to the pattern nodes are denoted by plus signs.

many details of the including trees can be omitted by describing them using a single pattern; hence the use of tree inclusion as a query language primitive provides some degree of data independence.

A tree U is a *minimal including tree of* P, if U includes P but no subtree of U does. The problem of locating subtrees of the target that are minimal including trees of the pattern is called the *tree inclusion problem* [13].

Gonnet and Tompa [7] have described an elegant grammatical model and query language for text databases. Following this work, Mannila and Räihä [17] proposed tree inclusion as a primitive for expressing the operations of the query language of Gonnet and Tompa. The tree inclusion problem has been shown solvable in $O(mn)$ time and space by Kilpeläinen and Mannila [13]. A query language and its implementation based on the concept of tree inclusion is described in [14].

Tree pattern matching and tree inclusion both seem to require superlinear time in the general case. In this paper we consider situations where the target tree represents a structured text, i.e., the target trees are parse trees over the grammar G defining the structure of the text.

If G is an arbitrary grammar, then the target tree can have an arbitrary structure. However, for structured text databases the grammars usually have a specific structure. Namely, iteration is expressed by using regular expressions in the right-hand-sides of productions, and no other means for recursion or iteration is used. We call such grammars *nonperiodic*. Note that nonperiodic grammars define exactly the regular languages, and that the height of a nonperiodic parse tree is bounded by the number of nonterminals in the grammar.

In this paper we show that tree pattern matching and tree inclusion can be solved in linear time for target trees that are parse trees over a nonperiodic grammar. For tree pattern matching the result is quite obvious, as even the trivial algorithm works in linear time for trees with nonperiodic underlaying grammars. For tree inclusion, the result is not so easy. We give these results using a general sufficient condition for linear solvability of tree matching problems; the condition can also be used to derive a couple of recent linearity results [16, 5, 8].

The rest of the paper is organized as follows. In Section 2 we define *general tree matching* and its special cases tree pattern matching and tree inclusion. We give a sufficient condition for general tree matching problems to be solvable in linear time. Then in Section 3 we define *grammatical* and *nonperiodic* tree matching problems and show that nonperiodic tree pattern matching is solvable in linear time. In Section 4 we show that nonperiodic tree inclusion problem is also solvable in time $O(n)$. Section 5 discusses possibilities to preprocess the pattern when the grammar of the target is known.

2 General Tree Matching

The *general tree matching* problem is to locate subtrees of the target that are *instances* of the pattern. A specific matching problem is defined by fixing the *instance relation* that specifies which trees are instances of which patterns. We say that the pattern *matches* at the root of the subtrees of the target that are instances of the pattern.

The instance relation of tree pattern matching is the relation between trees and their pruned counterparts: Tree P is a *pruned tree* of a tree U, if P can be obtained from U by deleting all descendants of zero or more nodes.

Problem 1 (Tree pattern matching). Given a pattern tree P and a target tree T, locate the subtrees U of T for which P is a pruned tree of U. □

The tree inclusion problem is defined via embeddings between trees. An injective function f from the nodes of P into the nodes of T is an *embedding* of P into T, if it preserves labels, ancestorship, and left-to-right-order, that is for all nodes u and v of pattern P we have

1. $label(u) = label(f(u))$,
2. u is an ancestor of v in P if and only if $f(u)$ is an ancestor of $f(v)$ in T, and
3. u precedes v in the postorder of P if and only if $f(u)$ precedes $f(v)$ in the postorder of T.

An embedding f of P into T is *root preserving* if $f(root(P)) = root(T)$.

Tree T *includes* P, or P is an *included tree* of T, if there is an embedding of P into T. A tree U is a *minimal including tree of* P, if there is an embedding of P into U and every embedding of P into U is root preserving.

Problem 2 (Tree inclusion problem). Given a pattern tree P and a target tree T, locate the subtrees of T that are minimal including trees of P. □

An instance relation is *linearly solvable*, if there is a constant c such that the question "Is U an instance of P?" can be answered in time bounded by $c|U|$ for all trees P and U. The relation "P and U are isomorphic *unordered* trees" is an example of a nontrivial linearly solvable instance relation [1, p. 84–86].

If P matches at a node v of T, we say that v is an *occurrence* of P. A set of nodes of T is a *candidate set of occurrences* of P, if it is superset of the set of occurrences of P. A set of nodes N is *k-thin*, if any node $n \in N$ has at most $k-1$ proper ancestors in N. A 1-thin set of nodes is *flat*. That is, a flat set of nodes does not contain two nodes one of which is an ancestor of the other. Note that in tree pattern matching there need not be a flat set of occurrences, since the pattern may match both at a node and at some of its descendants. The following lemma is almost immediate.

Lemma 3. Assume that for a tree matching problem a k-thin candidate set of occurrences can be computed in time $O(kn)$, and that the instance relation is linearly solvable. Then the tree matching problem is solvable in time $O(kn)$.

Proof. First compute a k-thin candidate set C in time $O(kn)$. For each node u in C, test whether P matches at u. Since the instance relation is linearly solvable, this requires time at most $c\sum_{u\in C}|tree(u)|$ for some constant c. Because C is k-thin, each node of T can belong to at most k trees rooted by nodes in C, and therefore

$$c\sum_{u\in C}|tree(u)| \le ckn = O(kn) \ .$$

□

Lemma 3 gives us an easy way to show the linearity of some tree matching problems. For example, the following result is immediate.

Corollary 4. Locating subtrees of the target that are isomorphic to the pattern as unordered trees can be done in time $O(n)$.

Proof. A flat candidate set of occurrences can be formed in time $O(n)$ by traversing the target bottom-up and counting the sizes of its subtrees. The roots of the subtrees of size m form a flat candidate set of occurrences. As stated above, the instance relation is linearly solvable. The result follows from Lemma 3. $\square$

The idea of Lemma 3 was used by Grossi in [8] for showing that locating subtrees of the target that are identical to the pattern or differ only with regard to don't care labels or up to m mismatching labels can be done in $O(n)$ sequential time.

3 Grammatical Tree Matching

Next we consider describing the structure of text databases by context-free grammars. We define a *grammar* to be a quadruple $G = (\mathcal{V}, \mathcal{T}, \mathcal{P}, S)$, where $\mathcal{V}$ is the set of *nonterminals*, $\mathcal{T}$ is the set of *terminals*, $\mathcal{P}$ is the set of *productions* and S is the *start symbol*.

It is useful to allow regular expressions on the right-hand-sides of productions. This leads to fewer nonterminals and seems to be a form easily comprehensible also to nonspecialists. Therefore, we define the productions to be of the form $A \rightarrow \alpha$, where α is a regular expression over $\mathcal{V} \cup \mathcal{T}$. We say that a production $A \rightarrow w$ is an *instance* of $A \rightarrow \alpha$, if w belongs to the regular language defined by α.

As an example we show a grammar for describing the structure of a list of bibliographic references stored in a text database system.

publications	$\rightarrow$	publication*
publication	$\rightarrow$	authors title journal volume year pages
authors	$\rightarrow$	author*
author	$\rightarrow$	initials name
initials	$\rightarrow$	text
name	$\rightarrow$	text
title	$\rightarrow$	text
journal	$\rightarrow$	text
volume	$\rightarrow$	number
year	$\rightarrow$	number
pages	$\rightarrow$	start end
start	$\rightarrow$	number
end	$\rightarrow$	number
text	$\rightarrow$	character*
number	$\rightarrow$	digit*

The obvious productions for nonterminals *character* and *digit* have been excluded. The grammar is allowed to be ambiguous, since we do not use it for parsing. Producing string representations out of a database and parsing strings into database instances can be performed by using versions of the grammar that are *annotated* with extra terminal symbols. The methodology is explained in [11] and in [18].

To define *parse trees* over a grammar G, we define sets $T(G, a)$ for terminals $a \in \mathcal{T}$ and sets $T(G, A)$ for nonterminals $A \in \mathcal{V}$.

$$T(G,a) = \{a\} \text{ for terminals } a \ ;$$

$$\begin{aligned} T(G,A) = \{A(t_1,\ldots,t_n) \mid\ & A \to B_1,\ldots,B_n \\ & \text{is an instance of a production in } P \\ & \text{and } t_i \in T(G,B_i) \text{ for each } i = 1,\ldots,n\} \ ; \end{aligned}$$

Here $A(t_1,\ldots,t_n)$ stands for a tree whose root is labeled by the nonterminal A and whose ith immediate subtree is t_i for all $i = 1,\ldots,n$. That is, elements of $T(G,A)$ represent derivations of terminal strings from the nonterminal A according to G. Finally, the trees that represent derivations from the start symbol of G, i.e., the trees in $T(G,S)$, are the parse trees over G.

A tree matching problem is *G-grammatical*, if the target is a parse tree over a grammar G. Grammatical matching problems are in general no easier than the unrestricted ones, since a grammatical problem for the grammar

$$\begin{aligned} &S \to a_1 \mid a_2 \mid \ldots \mid a_l \\ &a_1 \to (a_1 \mid a_2 \mid \ldots \mid a_l)^* \\ &\quad\vdots \\ &a_l \to (a_1 \mid a_2 \mid \ldots \mid a_l)^* \ , \end{aligned}$$

where $\{a_1,\ldots,a_l\}$ is the set of labels, is the same as the problem for arbitrary trees.

A grammar is *nonperiodic*[2] if it has no nonterminal A that can derive a string of the form $\alpha A \beta$. A tree T is *k-periodic* if any nonterminal appears at most k times on a single root-to-leaf path in T. A 1-periodic tree is *nonperiodic*, or equivalently, a tree T is nonperiodic if and only if T is a parse tree over some nonperiodic grammar. A matching problem (P,T) is *nonperiodic*, if it is G-grammatical for a nonperiodic grammar G (i.e., the target T is nonperiodic).

Although nonperiodic grammars are too weak for modeling programming languages, we argue that they are powerful enough to model the structure of most text databases. Recall that we allow regular expressions in the productions of the grammar. Therefore, a language defined by a nonperiodic grammar could be defined by giving one regular expression. For example, we could describe the previous list of bibliographic references also by a single production of the following form.

$$\begin{aligned} \text{publications} \quad \to \quad & ((\text{character}^* \ \text{character}^*)^* \\ & \text{character}^* \ \text{character}^* \\ & \text{digit}^* \ \text{digit}^* \ \text{digit}^* \ \text{digit}^*)^* \end{aligned}$$

The subexpressions *character* and *digit* that simply describe the set of recognized characters and the digits '0'–'9' have been left unspecified. It should be clear from this example that a single regular expression is not a convenient description for the

[2] Nonperiodic grammars are usually called *nonrecursive*. We use the term nonperiodic to avoid confusing nonperiodic matching problems with nonrecursive, i.e., undecidable problems.

logical structure of a text database. In practice, nonperiodic grammars with regular expressions in productions support modeling long lists of, say, dictionary articles, but unlimited nesting of structures is of course not possible.

In many applications the grammar G describing the structure of the text is fairly small. However, this is not always the case. For example, in a dictionary for Finnish the first third of the dictionary (about 17 000 words) has 1300 different structures for the entries, and there does not seem to be any simple description of this structure using a small grammar. Still, the parse tree is 2-periodic, and it can be simply transformed to a nonperiodic form.

Note that if T is a parse tree over a nonperiodic grammar G, then the height of T is at most $|\mathcal{V}| + 1$. It is known that restricting the height of the pattern improves the running time of the trivial algorithm for tree pattern matching:

Lemma 5. [4] If the height of the pattern is h, then the trivial algorithm for tree pattern matching takes time $O(nh)$. □

In a matching problem the height of P is at most the height of T. Thus nonperiodic tree pattern matching can be solved using the trivial algorithm in time $O(|\mathcal{V}|\, n)$. Next we show how Lemma 3 makes it possible to solve the tree pattern matching problem in time $O(kn)$ for a k-periodic target, and hence in time $O(n)$ for an arbitrary nonperiodic grammar G.

Lemma 6. A k-thin candidate set of occurrences for tree pattern matching and for tree inclusion can be computed in a k-periodic target in time $O(n)$.

Proof. The nodes of the target which have the same label as the root of the pattern form a candidate set of occurrences; they can be located by a simple traversal of the target. The set is k-flat, because the target is k-periodic. □

Lemma 7. The instance relation "P is a pruned tree of U" is linearly solvable.

Proof. The relation can be tested simply by comparing the corresponding nodes of the trees against each other; at most $\min\{|P|, |U|\}$ nodes of U are examined. □

Theorem 8. Tree pattern matching with a k-periodic target can be solved in time $O(kn)$, and the nonperiodic tree pattern matching can be solved in time $O(n)$.

Proof. Follows directly from Lemmas 3, 6, and 7. □

Theorem 8 shows that nonperiodic tree pattern matching is solvable in linear time. To obtain the same result for nonperiodic tree inclusion, we show in the next section that the instance relation "U is a minimal including tree of P" is solvable in linear time, when U is nonperiodic.

4 Solving Nonperiodic Tree Inclusion

Testing for the existence of a root preserving embedding of P into U amounts to searching appropriately related embeddings of the subtrees of P into the subtrees of

T. There may be exponentially many ways to embed the subtrees of P in T, but it was shown in [13] that this complexity can be avoided by concentrating on so called left embeddings.

A *forest* is an ordered sequence of trees $\langle T_1, \ldots, T_k \rangle$, $k \geq 0$. An embedding of a forest into another one is defined in the same way as an embedding between trees. Let P be a tree with immediate subtrees $\langle P_1, \ldots, P_k \rangle$ and U a tree with immediate subtrees $\langle U_1, \ldots, U_l \rangle$. It is obvious that there is a root preserving embedding of P into U if and only if $label(root(P)) = label(root(U))$ and there is an embedding of $\langle P_1, \ldots, P_k \rangle$ into $\langle U_1, \ldots, U_l \rangle$.

We denote the preorder and postorder numbers of a node n by $pre(n)$ and $post(n)$. In order to discuss the order of images of nodes in embeddings we define the *right relatives* of a node. Let F be a forest, N the set of its nodes, and u a node in F. The set of right relatives of u is defined by

$$rr(u) = \{x \in N \mid pre(u) < pre(x) \ \wedge \ post(u) < post(x)\} \ ,$$

i.e., the right relatives of u are those nodes that follow u both in preorder and in postorder. (See Fig. 3.)

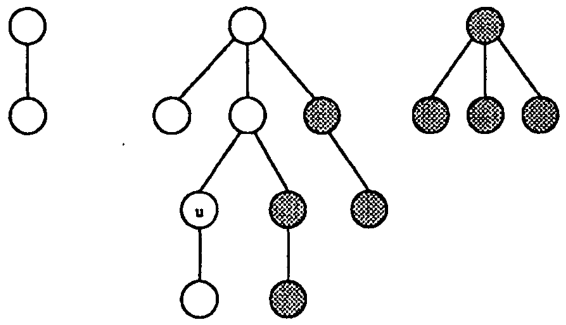

Fig. 3. The right relatives of the node u.

The following lemma tells us how to construct an embedding of a forest out of the embeddings of its trees.

Lemma 9. [13] Let $F = \langle T_1, \ldots, T_k \rangle$ and G be forests. There is an embedding of F into G if and only if for every $i = 1, \ldots, k$ there is an embedding f_i of T_i into G such that $f_{i+1}(root(T_{i+1}))$ is a right relative of $f_i(root(T_i))$, whenever $1 \leq i < k$. □

Let $F = \langle T_1, \ldots, T_n \rangle$ and G be forests. An embedding f of F into G is a *left embedding* of F into G if for every embedding g of F into G

$$post(f(root(T_i))) \leq post(g(root(T_i))), \ i = 1, \ldots, n \ .$$

Theorem 10. [13] Let F and G be forests. There is an embedding of F into G if and only if there is a left embedding of F into G. □

Next we give an algorithm for testing the instance relation of nonperiodic tree inclusion between a pattern P and a tree U. Denote the set of proper descendants of a node v by $desc(v)$. The nonperiodicity of U means that if v is a node in U then there are no nodes labeled by $label(v)$ in $desc(v)$. This implies two further facts utilized in the algorithm. First, tree U is a minimal including tree of P if and only if there is a root preserving embedding of P into U. Second, let N be a nonempty set of nodes of U all of which have the same label. Then the first node of N in the preorder of U and the first node of N in the postorder of U are the same node.

Let $u_1, \ldots, u_k$ be the children of the root of P, and $P_1, \ldots, P_k$ the corresponding immediate subtrees of P. The algorithm uses a pointer p for traversing the descendants of the root of U. First our algorithm searches for the image $f(u_1)$ under a left embedding f of $\langle P_1 \rangle$ into the forest of immediate subtrees of U, if one exists. After finding a left embedding f for the forest $\langle P_1, \ldots, P_i \rangle$, the pointer p points at the node $f(u_i)$. In order to extend f to a left embedding of $\langle P_1, \ldots, P_{i+1} \rangle$ we search the closest right relative x of p in U, such that there is a root preserving embedding of P_{i+1} into $tree(x)$.

Algorithm 11. Testing the instance relation of nonperiodic tree inclusion.

Input: Trees P and U, where U is nonperiodic, and nodes u and v, where $u = root(P)$ and $v = root(U)$.
Output: **true** if and only if U is a minimal including tree of P.
Method: If $label(u) = label(v)$, call emb(u, v); otherwise return **false**

```
1. function emb($u, v$);
2.    if $u$ is a leaf then return true;
3.    else Let $u_1, \ldots, u_k$ be the children of $u$;
4.         Let $p$ be the first descendant of $v$ in preorder
           satisfying $label(p) = label(u_1)$;
           if there is no such node $p$ then return false; fi;
5.         $i := 1$;
6.         while $i \le k$ do
7.              if emb$(u_i, p)$ then $i := i + 1$; fi;
8.              if $i \le k$ then
9.                   Let $p$ be the first node in $rr(p) \cap desc(v)$ in
                     preorder of $U$ satisfying $label(p) = label(u_i)$;
                     if there is no such $p$ then return false; fi;
10.             fi;
11.        od;
12.        return true;
13.   fi;
14. end;
```

Lemma 12. Algorithm 11 tests the relation "U is a minimal including tree of P" correctly for all trees P and all nonperiodic trees U.

Proof. If P consists of a single node u, the claim is obvious. Then assume that the height of P is $h > 0$ and and that the algorithm works correctly on all patterns of height less than h. Let the immediate subtrees of P rooted by $u_1, \ldots, u_k$ be $P_1, \ldots, P_k$. Now the following two invariants can be shown to hold for the loop on lines 6–11. First, before each execution of the loop, $post(f(u_i)) \geq post(p)$ for all embeddings f of $\langle P_1, \ldots, P_i \rangle$ into the forest of immediate subtrees of U. Second, after each execution of the loop, the forest $\langle P_1, \ldots, P_{i-1} \rangle$ has a left embedding into the forest of immediate subtrees of U. The correctness of the algorithm follows from these invariants. □

Lemma 13. The instance relation "U is a minimal including tree of P" is linearly solvable for nonperiodic trees U.

Proof. Algorithm 11 tests the relation for nonperiodic trees U correctly by Lemma 12. We show that there is a constant c such that the algorithm works in time bounded by $c|U|$ for all trees P and U.

Denote by $t(n)$ the maximum time needed to compare the root labels of P and U and perform the function call emb($root(P)$, $root(U)$), when P and U are trees and $|U| = n$. First, consider testing a single node. Obviously there is a constant c' such that $t(1) \leq c'$. Then assume that $|U| = n > 1$ and $t(m) \leq c'm$ for all targets of size $m < n$. Let n'' be the number of nodes of U that are examined during the traversal on lines 4 and 9 of the algorithm, excluding the roots of the subtrees of U that are examined in the recursive calls on line 7. Let n' be the total size of the subtrees of U that are examined in the recursive calls. There is a constant c'' such that the traversal can be performed in time $c''n''$. Therefore for $n > 1$ we get $t(n) \leq c'n' + c''n''$. Since the search proceeds in preorder, and the subtrees examined in the recursive calls do not overlap, $n' + n'' \leq n$. Hence for all $n > 0$ we have $t(n) \leq cn$ by selecting $c = \max\{c', c''\}$. □

Theorem 14. Nonperiodic tree inclusion problem can be solved in $O(n)$ time.

Proof. By Lemma 6 a flat candidate set of occurrences can be computed in $O(n)$ time, and by Lemma 13 the instance relation is linearly solvable. Therefore, by Lemma 3 the problem is solvable in linear total time. □

Periodic targets seem to be more difficult in the case of tree inclusion. Trying to apply the approach of Algorithm 11 to testing the instance relation with k-periodic targets seems to lead to $\Omega(2^k n)$ worst case running times. This means that the specialized approach of this paper would be for k-periodic targets more efficient than the general $O(mn)$ tree inclusion algorithm of [13] if $k < \log m$.

5 Preprocessing Grammatical Patterns

In the nonperiodic tree matching problems considered above obviously only nonperiodic patterns can have occurrences in the target. More generally, in a G-grammatical matching problem one can check, before performing the actual matching, whether P can have an instance in a parse tree over G. For example, in a text database application patterns not passing this test could result in informative diagnostics about the impossibility of locating data in the database using such patterns. Such preprocessing is probably useful only when the grammar of the database is essentially smaller than the parse tree of the database.

In what follows, we can assume that the grammar of the target $G = (\mathcal{V}, \mathcal{T}, \mathcal{P}, S)$ contains only useful nonterminals. That is, every nonterminal in $\mathcal{V}$ appears in a derivation of a string of terminals from the start symbol S of G. (See [10, p. 88–90].)

For tree pattern matching one checks that each node of pattern P is labeled by a symbol of grammar G and that the children of each internal node of P correspond to an appropriate production in G. Let u be an internal node of P with label A and with children $u_1, \ldots, u_k$. Checking u is performed by applying to the string of labels in $u_1, \ldots, u_k$ an automaton that recognizes the instances of the right-hand sides of the productions for nonterminal A. These finite automata need to be constructed only once for a grammar. If the automata are deterministic, checking pattern P takes only time $O(|P|)$. If the worst case $O(2^{|G|})$ size of the deterministic automata is prohibitive, it is also possible to construct nondeterministic finite automata for the same task; this can be done in time $O(|G|)$. The NFAs can be simulated for checking the children of each pattern node yielding total time $O(|G||P|)$. (See [1].)

For the tree inclusion problem the check is slightly more complicated. For each node of P labeled by A and having children labeled by $a_1, \ldots, a_k$ the following should hold in G:

$$A \overset{*}{\Rightarrow} \beta_0 a_1 \beta_1 \ldots \beta_{k-1} a_k \beta_k \ ,$$

where $\beta_i \in (\mathcal{V} \cup \mathcal{T})^*$. This can be checked in the following manner. For each nonterminal $B \in \mathcal{V}$ let B' be a unique terminal not belonging to $\mathcal{T}$, and for a set of nonterminals N let $N' = \{A' \mid A \in N\}$. For each terminal $t \in \mathcal{T}$ let t' be a unique new nonterminal not belonging to $\mathcal{V}$, and for a set of terminals C let $C' = \{t' \mid t \in C\}$. For a production p denote by p' the production obtained by replacing each terminal t in p by t'. For a set of productions Q denote by Q' the set $\{p' \mid p \in Q\}$. Finally, for grammar $G = (\mathcal{V}, \mathcal{T}, \mathcal{P}, S)$ and a nonterminal $A \in \mathcal{V}$ let $G'_A = (\mathcal{V}'', \mathcal{T}'', \mathcal{P}'', S'')$ be the grammar with

$$\begin{aligned}
\mathcal{V}'' &= \mathcal{V} \cup \mathcal{T}' \\
\mathcal{T}'' &= \mathcal{T} \cup \mathcal{V}' \\
\mathcal{P}'' &= P' \cup \\
&\qquad \{A \rightarrow (A' \mid \epsilon) \mid A \in \mathcal{V}\} \cup \\
&\qquad \{t' \rightarrow (t \mid \epsilon) \mid t \in \mathcal{T}\} \\
S'' &= A \ .
\end{aligned}$$

The idea is that grammar G'_A generates the subsequences of the sentential forms that are derivable from nonterminal A in grammar G. (Note that G'_A and G'_B may differ only with regard to the start symbol.) Now checking a pattern node labeled by A and having children labeled by $a_1, \ldots, a_k$ is done by first substituting B' for each nonterminal B in the sequence $a_1, \ldots, a_k$ and then parsing this sequence using G'_A.

We have above outlined possibilities to check the patterns against the grammar before performing a grammatical tree matching. Another promising direction for preprocessing patterns with regard to the grammar, analogical to query optimization in databases, is trying to transform the given matching problem to an easier one that still yields the same set of occurrences as the original problem.

For some patterns P and grammars G we may be able to compute a unique sequence of labels on any path between two nodes labeled by a and b in any parse tree over G, when a and b are labels of a node u and its descendant v in pattern P. Such knowledge allows us to complete the tree inclusion pattern by adding nodes labeled by the corresponding sequence of labels between every node-descendant pair u, v in P, and to solve the problem, possibly more efficiently, as a variant of tree pattern matching. Another problem that is feasible for this transformation is *unordered* tree inclusion. In this variation of the problem the order of the subtrees is not significant. It would sometimes be convenient to express queries on a grammatical database using tree inclusion but ignoring the left-to-right order of subtrees. Unfortunately, unordered tree inclusion is an NP-complete problem [12]. If the unordered tree inclusion pattern P and grammar G allow transforming P as above, the problem reduces to a subgraph isomorphism problem between trees. The latter problem is solvable in polynomial time [20].

6 Conclusion

We have shown that the restriction to nonperiodic targets makes two tree matching problems solvable in linear time. While the nonperiodicity assumption makes tree pattern matching and tree inclusion easier, it does not always help. For example, unordered tree inclusion is NP-complete, and remains NP-complete even for nonperiodic targets [13]. Similarly, adding logical variables to tree inclusion makes the problem NP-complete, and the restriction to nonperiodic targets does not help [14].

The restriction to nonperiodic targets does not seem severe for text database applications, but can be problematic in other applications, for example in code generation. We have shown how one can obtain tree pattern matching algorithms whose running times are proportional to the amount of periodicity in the target.

We have also outlined possibilities to check and preprocess patterns with regard to the grammar in grammatical tree matching problems.

In practical applications queries accessing structured texts contain in most cases a string search component. An interesting topic for further research is how one could integrate efficient string search mechanisms and tree inclusion algorithms. A simple starting point would be to use string searches to locate a set of candidate occurrences, and then find the real occurrences using a tree inclusion algorithm.

Acknowledgements

We wish to thank Jyrki Kivinen for his comments on the previous version of this paper.

References

1. A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, 1974.

2. F. Bancilhon and P. Richard. Managing texts and facts in a mixed data base environment. In G. Gardarin and E. Gelenbe, editors, *New Applications of Data Bases*. Academic Press, 1984.
3. G. Coray, R. Ingold, and C. Vanoirbeek. Formatting structured documents: Batch versus interactive. In J.C. van Vliet, editor, *Text Processing and Document Manipulation*. Cambridge University Press, 1986.
4. M. Dubiner, Z. Galil, and E. Magen. Faster tree pattern matching. In *Proc. of the Symposium on Foundations of Computer Science (FOCS'90)*, pages 145–150, 1990.
5. P. Dublish. Some comments on the subtree isomorphism problem for ordered trees. *Information Processing Letters*, 36:273–275, 1990.
6. R. Furuta, V. Quint, and J. André. Interactively editing structured documents. *Electronic Publishing*, 1(1):19–44, 1988.
7. G. H. Gonnet and F. Wm. Tompa. Mind your grammar - a new approach to text databases. In *Proc. of the Conference on Very Large Data Bases (VLDB'87)*, pages 339–346, 1987.
8. R. Grossi. A note on the subtree isomorphism for ordered trees and related problems. *Information Processing Letters*, 39:81–84, 1991.
9. C. M. Hoffman and M. J. O'Donnell. Pattern matching in trees. *Journal of the ACM*, 29(1):68–95, January 1982.
10. J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.
11. P. Kilpeläinen, G. Lindén, H. Mannila, and E. Nikunen. A structured document database system. In Richard Furuta, editor, *EP90 – Proceedings of the International Conference on Electronic Publishing, Document Manipulation & Typography*, The Cambridge Series on Electronic Publishing. Cambridge University Press, 1990.
12. P. Kilpeläinen and H. Mannila. Ordered and unordered tree inclusion. Report A-1991-4, University of Helsinki, Dept. of Comp. Science, August 1991.
13. P. Kilpeläinen and H. Mannila. The tree inclusion problem. In Samson Abramsky and T.S.E. Maibaum, editors, *TAPSOFT'91, Proc. of the International Joint Conference on the Theory and Practice of Software Development, Vol. 1: Colloqium on Trees in Algebra and Programming (CAAP'91)*, pages 202–214. Springer-Verlag, 1991.
14. P. Kilpeläinen and H. Mannila. A query language for structured text databases. Manuscript in preparation, February 1992.
15. S. R. Kosaraju. Efficient tree pattern matching. In *Proc. of the Symposium on Foundations of Computer Science (FOCS'89)*, pages 178–183, 1989.
16. E. Mäkinen. On the subtree isomorphism problem for ordered trees. *Information Processing Letters*, 32:271–273, September 1989.
17. H. Mannila and K.-J. Räihä. On query languages for the p-string data model. In H. Kangassalo, S. Ohsuga, and H. Jaakkola, editors, *Information Modelling and Knowledge Bases*, pages 469–482. IOS Press, 1990.
18. E. Nikunen. Views in structured text databases. Phil.lic. thesis, University of Helsinki, Department of Computer Science, December 1990.
19. V. Quint and I. Vatton. GRIF: An interactive system for structured document manipulation. In J.C. van Vliet, editor, *Proceedings of the International Conference on Text Processing and Document Manipulation*. Cambridge University Press, 1986.
20. S. W. Reyner. An analysis of a good algorithm for the subtree problem. *SIAM Journal of Computing*, 6(4):730–732, December 1977.

Organization TC2100 Bldg/Room RANDOLPH
United States Patent and Trademark Office
P.O. Box 1450
Alexandria, VA 22313-1450
If Undeliverable Return in Ten Days

OFFICIAL BUSINESS
PENALTY FOR PRIVATE USE, $300

AN EQUAL OPPORTUNITY EMPLOYER

STAT860 943032227 1307 38 07/02/08
FORWARD TIME EXP RTN TO SEND
STATTLER LAW CORPORATION
60 S MARKET ST STE 480
SAN JOSE CA 95113-2335

RETURN TO SENDER

RECEIVED
JUL 08 2008
USPTO MAIL CENTER